



*The Apple IIc  
Reference Manual  
Volume 1*

*The Apple IIc*



## **Customer Satisfaction**

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

In addition, if Apple releases a corrective update to a software product during the 90-day period after you purchased the software, Apple will replace the applicable disks and documentation with the revised version at no charge to you during the six months after the date of purchase.

In some countries the replacement period may be different; check with your authorized Apple dealer. Return any item to be replaced with proof of purchase to Apple or an authorized Apple dealer.

## **Limitation on Warranties and Liability**

Even though Apple has tested the software described in the manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is," and you the purchaser are assuming the entire risk as to their quality and performance. In no event will Apple or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

## **Copyright**

This manual and the software (computer programs) described in it are copyrighted by Apple or by Apple's software suppliers, with all rights reserved. Under the copyright laws, this manual or the programs may not be copied, in whole or part, without the written consent of Apple, except in the normal use of the software or to make a backup copy. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose. For some products, a multi-use license may be purchased to allow the software to be used on more than one computer owned by the purchaser, including a shared-disk system. (Contact your authorized Apple dealer for information on multi-use licenses.)

## **Product Revisions**

Apple cannot guarantee that you will receive notice of a revision to the software described in the manual, even if you have returned a registration card received with the product. You should periodically check with your authorized Apple dealer.

© Apple Computer, Inc. 1984  
20525 Mariani Avenue  
Cupertino, California 95014

Apple, the Apple logo, and ProDOS are trademarks of Apple Computer, Inc.  
Simultaneously published in the United States and Canada. All rights reserved.

## **Warning**

This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.



*Apple IIc Reference Manual*  
Volume 1

*The Apple IIc*



# *Table of Contents*



## ***Volume 1***

---

### ***List of Figures and Tables***

**xv**

---

### ***Preface***

**xxv**

---

### ***Chapter 1***

***Introduction*****1**

- 1 1.1 Outside of Machine
- 2 1.1.1 The Keyboard
- 3 Features
- 4 Special Function Keys
- 4 Cursor Movement Keys
- 4 Modifier Keys
- 5 The 80/40 Switch
- 5 The Keyboard Switch
- 6 Disk-Use and Power Lights
- 7 1.1.2 The Speaker
- 8 1.1.3 The Built-in Disk Drive
- 8 1.1.4 The Back Panel
- 10 1.2 Inside of Machine
- 10 1.2.1 The Internal Voltage Converter
- 11 1.2.2 The Main Logic Board
- 13 1.2.3 The Other Circuit Boards

15	2.1 The 65C02 Microprocessor
17	2.2 Overview of the Address Space
18	2.3 Memory Map and Memory Switching
20	2.3.1 Main RAM Addresses (\$0000-\$BFFF and \$D000-\$FFFF)
20	2.3.2 Auxiliary RAM Addresses (\$0000-\$BFFF and \$D000-\$FFFF)
20	2.3.3 ROM Addresses (\$C100-\$FFFF)
21	2.3.4 Hardware Addresses (\$C000-\$C0FF)
22	2.4 Bank-Switched Memory
24	2.4.1 Page Allocations
24	Page \$00 (One-Byte Addresses)
24	Page \$01 (The 65C02 Stack)
24	Pages \$D0-\$FF (ROM and RAM)
25	2.4.2 Using Bank Selector Switches
34	2.5 48K Memory
36	2.5.1 Page Allocations
36	Page \$02 (The Input Buffer)
36	Page \$03 (Global Storage and Vectors)
36	Pages \$04 Through \$07 (Text and Low-Resolution Page 1)
37	Pages \$08 Through \$0B (Text and Low-Resolution Page 2)
37	Page \$08 (Communication Port Buffers)
37	Pages \$20 Through \$3F (High-Resolution Page 1)
38	Pages \$40 Through \$5F (High-Resolution Page 2)
38	2.5.2 Using 48K Memory Switches
41	2.5.3 Transfers Between Main and Auxiliary Memory
41	Transferring Data
42	Transferring Control
43	2.5.4 Using Display Memory Switches
48	2.6 The Reset Routine
49	2.6.1 The Cold-Start Procedure (Power On)
50	2.6.2 The Warm-Start Procedure (CONTROL-RESET)
50	2.6.3 Forced Cold Start (¢-CONTROL-RESET)
51	2.6.4 The Reset Vector

55	3.1 The Standard I/O Links
56	3.1.1 Changing the Standard I/O Links
57	3.2 Standard Input Features
57	3.2.1 RDKEY Input Subroutine
58	3.2.2 KEYIN Input Subroutine
58	3.2.3 GETLN Input Subroutine
60	3.2.4 Escape Codes With GETLN
62	3.2.5 Editing With GETLN
62	Cancel Line
62	Backspace
62	Retype
63	3.3 Standard Output Features
63	3.3.1 COUT Output Subroutine
64	3.3.2 Control Characters With COUT1
64	3.3.3 Control Characters With C3COUT1
66	3.3.4 The Stop-List Feature
66	3.3.5 The Text Window
68	3.3.6 Normal, Inverse, and Flashing Text
68	Primary Character Set Display
69	Alternate Character Set Display
70	3.4 Port I/O
70	3.4.1 Standard Link Entry Points
71	3.4.2 Firmware Protocol
73	3.4.3 Port I/O Space
73	3.4.4 Port ROM Space
73	3.4.5 Expansion ROM Space
73	3.4.6 Port Screen-Hole RAM Space
75	3.5 Interrupts

77	4.1 Keyboard Input
79	4.1.1 Reading the Keyboard
82	4.1.2 Monitor Firmware Support
82	GETLNZ
82	GETLN1
82	RDCHAR
83	4.2 Speaker Output
83	4.2.1 Using the Speaker
84	4.2.2 Monitor Firmware Support
84	BELL1
84	BELL

**Chapter 5*****Video Display Output*****87**

- 89 5.1 Specifications
- 90 5.2 Text Modes
  - 90 5.2.1 Text Character Sets
  - 91 5.2.2 MouseText
  - 93 5.2.3 40-Column Versus 80-Column Text
- 96 5.3 Graphics Modes
  - 96 5.3.1 Low-Resolution Graphics
  - 97 5.3.2 High-Resolution Graphics
  - 100 5.3.3 Double-High-Resolution Graphics
- 102 5.4 Mixed-Mode Displays
- 102 5.5 Display Pages
- 104 5.6 Display Mode Switching
- 108 5.7 Display Page Maps
- 115 5.8 Monitor Firmware Support
- 120 5.9 I/O Firmware Support

**Chapter 6*****Disk Input and Output*****125**

- 126 6.1 Startup
- 126 6.2 External Drive Startup

**Chapter 7*****Serial I/O Port 1*****129**

- 130 7.1 Using Serial Port 1
- 133 7.2 Characteristics at Startup
- 134 7.3 Hardware Page Locations
- 134 7.4 I/O Firmware Support
- 135 7.5 Screen Hole Locations
- 136 7.6 Changing Port Characteristics
  - 137 7.6.1 Data Format and Baud Rate
  - 138 7.6.2 Carriage Return and Line Feed
  - 139 7.6.3 Sending Special Characters
  - 139 7.6.4 Displaying Output on the Screen

**Chapter 8****Serial I/O Port 2****141**

- 143 8.1 Using Serial Port 2
- 147 8.2 Characteristics at Startup
- 148 8.3 Hardware Page Locations
- 148 8.4 I/O Firmware Support
- 149 8.5 Screen Hole Locations
- 150 8.6 Changing Port Characteristics
  - 151 8.6.1 Data Format and Baud Rate
  - 152 8.6.2 Carriage Return and Line Feed
  - 153 8.6.3 Routing Input and Output
  - 155 Half Duplex Operation
  - 156 Full Duplex Operation
  - 159 Terminal Mode

**Chapter 9****Mouse and Game Input****161**

- 162 9.1 Mouse Input
  - 162 9.1.1 Mouse Connector Signals
  - 163 9.1.2 Mouse Operating Modes
    - 163 Transparent Mode
    - 163 Movement Interrupt Mode
    - 164 Button Interrupt Mode
    - 164 Movement/Button Interrupt Mode
    - 164 Vertical Blanking Active Modes
  - 164 9.1.3 Mouse Hardware Page Locations
  - 167 9.1.4 I/O Firmware Support
    - 170 Pascal Support
    - 171 BASIC and Assembly-Language Support
  - 171 9.1.5 Screen Holes
  - 173 9.1.6 Using the Mouse as a Hand Control
- 174 9.2 Game Input
  - 174 9.2.1 The Hand Control Signals
    - 175 Switch Inputs (SW0 and SW1)
    - 176 Analog Inputs (PDL0 and PDL1)
  - 177 9.2.2 Monitor Support
  - 177 PREAD



179	10.1 Invoking the Monitor
180	10.2 Syntax of Monitor Commands
181	10.3 Monitor Memory Commands
181	10.3.1 Examining Memory Contents
182	10.3.2 Memory Dump
184	10.3.3 Changing Memory Contents
184	Changing One Byte
185	Changing Consecutive Locations
186	10.3.4 Moving Data in Memory
188	10.3.5 Comparing Data in Memory
189	10.4 Monitor Register Commands
190	10.4.1 Changing Registers
190	10.4.2 Examining Registers
191	10.5 Miscellaneous Monitor Commands
191	10.5.1 Display Inverse and Normal
192	10.5.2 Back to BASIC
193	10.5.3 Redirecting Input and Output
193	10.5.4 Hexadecimal Arithmetic
194	10.6 Special Tricks With the Monitor
194	10.6.1 Multiple Command Lines
195	10.6.2 Filling Memory
196	10.6.3 Repeating Commands
197	10.6.4 Creating Your Own Commands
198	10.7 Machine-Language Programs
198	10.7.1 Running a Program
199	10.7.2 Disassembled Programs
201	10.8 Summary of Monitor Commands
201	Examining Memory
202	Changing the Contents of Memory
202	Moving and Comparing
202	The Register Command
202	Miscellaneous Monitor Commands
203	Running and Listing Programs

205	11.1 Environmental Specifications
206	11.2 Power Requirements
206	11.2.1 The External Power Supply
207	11.2.2 The External Power Connector
208	11.2.3 The Internal Converter

210	11.3 Apple IIc Overall Block Diagram
211	11.4 The CMOS 65C02 Microprocessor
212	11.4.1 65C02 Block Diagram
213	11.4.2 65C02 Timing
215	11.5 The Custom Integrated Circuits
216	11.5.1 The Memory Management Unit (MMU)
218	11.5.2 The Input/Output Unit (IOU)
220	11.5.3 The Timing Generator (TMG)
221	11.5.4 The General Logic Unit (GLU)
222	11.5.5 The Disk Controller Unit (IWM)
223	11.6 Memory Addressing
224	11.6.1 ROM Addressing
226	11.6.2 RAM Addressing
226	Dynamic-RAM Refreshment
227	Dynamic-RAM Timing
229	11.7 The Keyboard
232	11.8 The Speaker
232	11.8.1 Volume Control
232	11.8.2 Output Jack
233	11.9 The Video Display
233	11.9.1 The Video Counters
234	11.9.2 Display Memory Addressing
235	11.9.3 Display Address Mapping
239	11.9.4 Video Display Modes
241	Text Displays
242	Low-Resolution Display
243	High-Resolution Display
245	Double-High-Resolution Display
248	11.9.5 Video Output Signals
248	Monitor Output
249	Video Expansion Output
252	11.10 Disk I/O
253	11.11 Serial I/O
258	11.11.1 ACIA Control Register
260	11.11.2 ACIA Command Register
261	11.11.3 ACIA Status Register
262	11.11.4 ACIA Transmit/Receive Register
262	11.12 Mouse Input
267	11.13 Hand Controller Input
271	11.14 Schematic Diagrams

# Volume 2

<b>Appendix A</b>	<b><i>The 65C02 Microprocessor</i></b>	<b>1</b>
	2 A.1 Differences Between 6502 and 65C02	
	2 A.1.1 Differing Cycle Times	
	3 A.1.2 Differing Instruction Results	
	4 A.2 Data Sheet	
<b>Appendix B</b>	<b><i>Memory Map</i></b>	<b>15</b>
	15 B.1 Page Zero	
	19 B.2 Page Three	
	20 B.3 Screen Holes	
	23 B.4 The Hardware Page	
<b>Appendix C</b>	<b><i>Important Firmware Locations</i></b>	<b>31</b>
	31 C.1 The Tables	
	32 C.2 Port Addresses	
	34 C.3 Other Video and I/O Firmware Addresses	
	34 C.4 Applesoft BASIC Interpreter Addresses	
	34 C.5 Monitor Addresses	
<b>Appendix D</b>	<b><i>Operating Systems and Languages</i></b>	<b>37</b>
	37 D.1 Operating Systems	
	37 D.1.1 ProDOS	
	37 D.1.2 DOS	
	38 D.1.3 Pascal Operating System	
	38 D.1.4 CP/M	
	38 D.2 Languages	
	38 D.2.1 Applesoft BASIC	
	39 D.2.2 Integer BASIC	
	39 D.2.3 Pascal Language	
	39 D.2.4 FORTRAN	

41	E.1 Introduction
41	E.1.1 What Is an Interrupt
42	E.1.2 Interrupts on Apple II Computers
43	E.1.3 Interrupt Handling on the 65C02
43	E.1.4 The Interrupt Vector at \$FFFE
44	E.2 The Built-in Interrupt Handler
46	E.2.1 Saving the Memory Configuration
46	E.2.2 Managing Main and Auxiliary Stacks
47	E.3 User's Interrupt Handler at \$3FE
48	E.4 Handling Break Instructions
49	E.5 Sources of Interrupts
50	E.6 Firmware Handling of Interrupts
50	E.6.1 Firmware for Mouse and VBL
52	E.6.2 Firmware for Keyboard Interrupts
53	Using Keyboard Buffering Firmware
53	Using Keyboard Interrupts Through Software
54	E.6.3 Using External Interrupts Through Firmware
55	E.6.4 Firmware for Serial Interrupts
55	Using Serial Buffering Transparently
56	Using Serial Interrupts Through Firmware
57	Transmitting Serial Data
57	A Loophole in the Firmware
58	E.7 Bypassing the Interrupt Firmware
58	E.7.1 Using Mouse Interrupts Without the Firmware
59	E.7.2 Using ACIA Interrupts Without the Firmware

61	F.1 Overview
63	F.1.1 Type of CPU
63	F.1.2 Machine Identification
64	F.2 Memory Structure
64	F.2.1 Amount and Address Ranges of RAM
65	F.2.2 Amount and Address Ranges of ROM
66	F.2.3 Peripheral-Card Memory Spaces
66	F.2.4 Hardware Addresses
67	\$C000 to \$C00F
67	\$C010 to \$C01F
68	\$C020 to \$C02F
68	\$C030 to \$C03F
68	\$C040 to \$C04F
68	\$C050 to \$C05F
69	\$C060 to \$C06F
70	\$C070 to \$C07F
70	\$C080 to \$C08F
70	\$C090 to \$C0FF
71	F.2.5 Monitors
72	F.3 I/O in General
72	F.3.1 DMA Transfers
72	F.3.2 Slots Versus Ports
72	F.3.3 Interrupts
73	F.4 Keyboard
73	F.4.1 Keys
74	F.4.2 Character Sets
75	F.5 Speaker
75	F.6 Video Display
75	F.6.1 Character Sets
76	F.6.2 MouseText
76	F.6.3 Vertical Blanking
76	F.6.4 Display Modes
77	F.7 Disk I/O
77	F.8 Serial I/O
77	F.8.1 Serial Ports Versus Serial Cards
78	F.8.2 Serial I/O Buffers
79	F.9 Mouse and Hand Controls
79	F.9.1 Mouse Input
79	F.9.2 Hand Control Input and Output
80	F.10 Cassette I/O
81	F.11 Hardware
81	F.11.1 Power
81	F.11.2 Custom Chips



---

**Appendix G****USA and International Models****83**

- 83 G.1 Keyboard Layouts and Codes
- 85 G.1.1 USA Standard (Sholes) Keyboard
- 88 G.1.2 USA Simplified (Dvorak) Keyboard
- 89 G.1.3 ISO Layout of USA Keyboard
- 90 G.1.4 English Keyboard
- 91 G.1.5 French and Canadian Keyboards
- 93 G.1.6 German Keyboard
- 94 G.1.7 Italian Keyboard
- 96 G.1.8 Western Spanish Keyboard
- 97 G.2 ASCII Character Sets
- 99 G.3 Certifications
- 99 G.3.1 Radio Interference
- 99 G.3.2 Product Safety
- 99 G.3.3 Important Safety Instructions
- 100 G.4 Power Supply Specifications

---

**Appendix H****Conversion Tables****103**

- 103 H.1 Bits and Bytes
- 106 H.2 Hexadecimal and Decimal
- 107 H.3 Hexadecimal and Negative Decimal
- 109 H.4 Graphics Bits and Pieces
- 112 H.5 Peripheral Identification Numbers
- 114 H.6 Eight-Bit Code Conversions

---

**Appendix I****Firmware Listings****125**

---

**Glossary****219**

---

**Bibliography****243**

---

**Index****247**

---

**Tell Apple Card**

# *List of Figures and Tables*

# Volume 1

## Chapter 1

### Introduction

- 2 Figure 1-1 Apple IIc With Standard USA Keyboard
- 2 Figure 1-2 Back and Left Side of Computer
- 3 Figure 1-3 Front and Right Side of Computer
- 3 Table 1-1 Keyboard Specifications
- 5 Figure 1-4 The USA Standard or *Sholes* Keyboard (Keyboard Switch Up)
- 6 Figure 1-5 Simplified or *Dvorak* Keyboard (Keyboard Switch Down)
- 7 Figure 1-6 Speaker, Volume Control, and Phone Jack
- 8 Figure 1-7 Built-in Disk Drive
- 9 Figure 1-8 Back Panel Connectors
- 10 Figure 1-9 Block Diagram of Inside of Machine
- 11 Figure 1-10 Power Supply and Voltage Converter
- 12 Figure 1-11 Main Logic Board

## Chapter 2

### Memory Organization and Control

- 16 Figure 2-1 Block Diagram Model of 65C02
- 19 Figure 2-2 Apple IIc Memory Map
- 23 Figure 2-3 Bank-Switched Memory
- 26 Table 2-1 Bank Select Switches
- 27 Figure 2-4 Read ROM
- 28 Figure 2-5 Read ROM, Write RAM, and Use First \$D0 Bank
- 29 Figure 2-6 Read ROM, Write RAM, and Use Second \$D0 Bank
- 30 Figure 2-7 Read RAM and Use First \$D0 Bank
- 31 Figure 2-8 Read RAM and Use Second \$D0 Bank

32	Figure 2-9	Read and Write RAM and Use First \$D0 Bank
33	Figure 2-10	Read and Write RAM and Use Second \$D0 Bank
35	Figure 2-11	48K Memory Map
39	Table 2-2	48K Memory Switches
39	Figure 2-12	48K RAM Selection: Split Pairs
40	Figure 2-13	48K RAM Selection: One Side Only
41	Table 2-3	48K RAM Transfer Routines
42	Table 2-4	Parameters for MOVEAUX Routine
43	Table 2-5	Parameters for XFER Routine
45	Table 2-6	Display Memory Switches
46	Figure 2-14	PAGE2 Selections With 80STORE On and HIRES Off
47	Figure 2-15	PAGE2 Selections With 80STORE On and HIRES On
48	Figure 2-16	RESET Routine Flowchart
52	Table 2-7	Page 3 Vectors

## **Chapter 3**

### ***Introduction to Apple IIc I/O***

59	Table 3-1	Prompt Characters
60	Table 3-2	Escape Codes With GETLN
64	Table 3-3	Control Characters With COUT1
65	Table 3-4	Control Characters With C3COUT1
68	Table 3-5	Text Window Memory Locations
70	Table 3-6	Port Characteristics
72	Table 3-7	Firmware Protocol Locations
73	Table 3-8	Port I/O Locations
75	Table 3-9	Port Screen-Hole Locations

## **Chapter 4**

### ***Keyboard and Speaker***

77	Table 4-1	Keyboard Input Characteristics
80	Table 4-2	Keys and ASCII Codes
83	Table 4-3	Speaker Output Characteristics

## **Chapter 5**

### ***Video Display Output***

88	Table 5-1	Guide to the Information in This Chapter
89	Table 5-2	Video Display Specifications
91	Table 5-3	The Display Character Sets

92	Figure 5-1	MouseText Characters
94	Figure 5-2	40-Column and 80-Column Text (With Alternate Character Set)
95	Figure 5-3	Text Mode Characteristics and Switching
97	Table 5-4	Low-Resolution Graphics Colors
99	Table 5-5	High-Resolution Graphics Colors
100	Figure 5-4	High-Resolution Display Bits
101	Table 5-6	Double-High-Resolution Graphics Colors
103	Table 5-7	Video Display Page Locations
105	Table 5-8	Display Soft Switches
107	Table 5-9	Display Modes Supported by Firmware
107	Table 5-10	Other Display Modes
110	Figure 5-5	Map of 40-Column Text Display
111	Figure 5-6	Map of 80-Column Text Display
112	Figure 5-7	Map of Low-Resolution Graphics Display
113	Figure 5-8	Map of High-Resolution Graphics Display
114	Figure 5-9	Map of Double-High-Resolution Graphics Display
115	Table 5-11	Monitor Firmware Routines
120	Table 5-12	Port 3 Firmware Protocol Table
122	Table 5-13	Pascal Video Control Functions

## **Chapter 6**

### ***Disk Input and Output***

125	Table 6-1	Disk I/O Characteristics
-----	-----------	--------------------------

## **Chapter 7**

### ***Serial I/O Port 1***

130	Table 7-1	Serial Port 1 Characteristics
131	Table 7-2	Printer Port Commands
133	Table 7-3	Initial Characteristics of Printer Port
134	Table 7-4	Serial Port 1 Hardware Page Locations
134	Table 7-5	Port 1 I/O Firmware Protocol
135	Table 7-6	Port 1 Screen-Hole Locations
137	Figure 7-1	Port 1 Characteristics
138	Figure 7-2	Data Formats

## **Chapter 8**

### ***Serial I/O Port 2***

142	Table 8-1	Serial Port 2 Characteristics
144	Table 8-2	Modem Port Characteristics



147	Table 8-3	Initial Characteristics of Communication Port
148	Table 8-4	Serial Port 2 Hardware Page Locations
148	Table 8-5	Port 2 I/O Firmware Protocol
149	Table 8-6	Serial Port 2 Screen Hole Locations
151	Figure 8-1	Port 2 Characteristics
152	Figure 8-2	Devices in a Typical Communication Setup
154	Figure 8-3	Effect of IN#2
155	Figure 8-4	Effect of IN#2 and T Command (Half Duplex)
157	Figure 8-5	Effect of IN#2 and T Command (Full Duplex Terminal)
158	Figure 8-6	Effect of IN#2, PR#2 and T Command (Full Duplex Host)

## **Chapter 9**

### ***Mouse and Game Input***

162	Table 9-1	Mouse Input Port Characteristics
166	Table 9-2	Mouse Hardware Page Locations
168	Table 9-3	Mouse Firmware Routines
170	Table 9-4	Mouse Port I/O Firmware Protocol
172	Table 9-5	Mouse Peripheral Card RAM Locations
174	Table 9-6	Game Input Characteristics

## **Chapter 11**

### ***Hardware Implementation***

205	Table 11-1	Summary of Environmental Specifications
207	Table 11-2	Power Supply Specifications
207	Figure 11-1	External Power Connector
208	Table 11-3	External Power Connector Signals
208	Table 11-4	Internal Converter Specifications
210	Figure 11-2	Apple IIc Block Diagram
212	Figure 11-3	65C02 Block Diagram
213	Table 11-5	65C02 Microprocessor Specifications
214	Figure 11-4	65C02 Timing Signals
215	Table 11-6	65C02 Timing Signal Descriptions
217	Figure 11-5	The MMU Pinouts
217	Table 11-7	The MMU Signal Descriptions
218	Figure 11-6	The IOU Pinouts
218	Table 11-8	The IOU Signal Descriptions
220	Figure 11-7	The TMG Pinouts
220	Table 11-9	The TMG Signal Descriptions
221	Figure 11-8	The GLU Pinouts
221	Table 11-10	The GLU Signal Descriptions

222	Figure 11-9	The IWM Pinouts
222	Table 11-11	The IWM Signal Descriptions
224	Figure 11-10	Memory Bus Organization
225	Figure 11-11	The 23128 ROM Pinouts
225	Figure 11-12	The 2316 ROM Pinouts
225	Figure 11-13	The 2332/2364 ROM Pinouts
227	Figure 11-14	The 64K RAM Pinouts
227	Table 11-12	RAM Address Multiplexing
228	Figure 11-15	RAM Timing Signals
229	Table 11-13	RAM Timing Signals
230	Figure 11-16	Keyboard Circuit Diagram
231	Figure 11-17	Keyboard Signals
232	Figure 11-18	Speaker Circuit Diagram
236	Figure 11-19	Display Address Transformation
237	Figure 11-20	40-Column Text Display Memory
238	Table 11-14	Display Memory Addressing
238	Table 11-15	Memory Address Bits for Display Modes
240	Figure 11-21	Video Display Circuits
242	Table 11-16	Character-Generator Control Signals
246	Figure 11-22	7 MHz Video Timing Signals (40-Column, Low-Resolution and High-Resolution Display)
247	Figure 11-23	14 MHz Video Timing Signals (80-Column and Double-High-Resolution Display)
248	Figure 11-24	Video Output Back Panel Connectors
250	Figure 11-25	The Video Expansion Connector Pinouts
251	Table 11-17	The Video Expansion Connector Signals
252	Figure 11-26	Disk Drive Connector
253	Table 11-18	Disk Drive Connector Signals
254	Figure 11-27	Serial Port Circuits
255	Figure 11-28	6551 ACIA Block Diagram
256	Figure 11-29	The 6551 Pinouts
256	Table 11-19	The 6551 Signal Descriptions
257	Figure 11-30	Serial Port Connectors
257	Table 11-20	Serial Port Connector Signals
259	Figure 11-31	ACIA Control Register
260	Figure 11-32	ACIA Command Register
261	Figure 11-33	ACIA Status Register
263	Figure 11-34	Sample Mouse Waveform
263	Figure 11-35	Mouse Movement and Direction Waveforms

264	Figure 11-36	Mouse Connector
264	Table 11-21	Mouse Connector Signals
265	Figure 11-37	Mouse Circuits
266	Figure 11-38	Mouse Button Signals
267	Figure 11-39	Hand Controller Connector
268	Table 11-22	Hand Control Connector Signals
268	Figure 11-40	How to Connect Switch Inputs
269	Figure 11-41	Hand Control Circuits
270	Figure 11-42	Hand Control Signals

## Volume 2

### Appendix A

#### *The 65C02 Microprocessor*

2	Table A-1	Cycle Time Differences
---	-----------	------------------------

### Appendix B

#### *Memory Map*

16	Table B-1	Zero Page Use
19	Table B-2	Page 3 Use
20	Table B-3	Main Memory Screen Hole Allocations
22	Table B-4	Auxiliary Memory Screen Hole Allocations
24	Table B-5	Addresses \$C000 through \$C03F
25	Table B-6	Addresses \$C040 through \$C05F
26	Table B-7	Addresses \$C060 through \$C07F
27	Table B-8	Addresses \$C080 through \$C0AF
28	Table B-9	Addresses \$C0B0 through \$C0FF

### Appendix C

#### *Important Firmware Locations*

32	Table C-1	Serial Port 1 Addresses
32	Table C-2	Serial Port 2 Addresses
33	Table C-3	Video Firmware Addresses
33	Table C-4	Mouse Port Addresses
34	Table C-5	Apple IIc Enhanced Video and Miscellaneous Firmware
35	Table C-6	Apple IIc Monitor Entry Points and Vectors

---

**Appendix E****Interrupts**

- |    |            |                             |
|----|------------|-----------------------------|
| 45 | Figure E-1 | Interrupt-Handling Sequence |
| 58 | Table E-1  | Activating Mouse Interrupts |
| 58 | Table E-2  | Reading Mouse Interrupts    |

---

**Appendix F****Apple II Series Differences**

- |    |            |   |
|----|------------|---|
| 80 | Figure F-1 | Apple II, II Plus, and IIe Hand Control Signals |
|----|------------|---|

---

**Appendix G****USA and International Models**

- |     |            |   |
|-----|------------|---|
| 85  | Figure G-1 | USA Standard or <i>Sholes</i> Keyboard (Keyboard Switch Up)     |
| 86  | Table G-1  | Keys and ASCII Codes  |
| 88  | Figure G-2 | USA Simplified or <i>Dvorak</i> Keyboard (Keyboard Switch Down) |
| 89  | Figure G-3 | ISO Version of USA Standard Keyboard (Keyboard Switch Up)       |
| 90  | Figure G-4 | English Keyboard (Keyboard Switch Down)                         |
| 90  | Table G-2  | English Keyboard Code Differences From Table G-1                |
| 91  | Figure G-5 | French Keyboard (Keyboard Switch Down)                          |
| 92  | Figure G-6 | Canadian Keyboard (Keyboard Switch Down)                        |
| 92  | Table G-3  | French and Canadian Keyboard Code Differences From Table G-1    |
| 93  | Figure G-7 | German Keyboard (Keyboard Switch Down)                          |
| 93  | Table G-4  | German Keyboard Code Differences From Table G-1                 |
| 94  | Figure G-8 | Italian Keyboard (Keyboard Switch Down)                         |
| 95  | Table G-5  | Italian Keyboard Code Differences From Table G-1                |
| 96  | Figure G-9 | Western Spanish Keyboard (Keyboard Switch Down)                 |
| 96  | Table G-6  | Western Spanish Keyboard Code Differences From Table G-1        |
| 98  | Table G-7  | ASCII Code Equivalents  |
| 100 | Table G-8  | 50 Hz Power Supply Specifications                               |

104	Table H-1	What a Bit Can Represent
105	Figure H-1	Bits, Nibbles, and Bytes
106	Table H-2	Hexadecimal/Decimal Conversion
108	Table H-3	Decimal to Negative Decimal Conversion
109	Table H-4	Hexadecimal Values for High-Resolution Dot Patterns
113	Table H-5	PIN Numbers
115	Table H-6	Control Characters, High Bit Off
116	Table H-7	Special Characters, High Bit Off
117	Table H-8	Uppercase Characters, High Bit Off
118	Table H-9	Lowercase Characters, High Bit Off
119	Table H-10	Control Characters, High Bit On
120	Table H-11	Special Characters, High Bit On
121	Table H-12	Uppercase Characters, High Bit On
122	Table H-13	Lowercase Characters, High Bit On



## Radio Frequency Interference Statement

The equipment described in this manual generates and uses radio-frequency energy. If it is not installed and used properly, that is, in strict accordance with our instructions, it may cause interference with radio and television reception.

This equipment has been tested and complies with the limits for a Class B computing device in accordance with the specifications in Subpart J, Part 15, of FCC rules. These rules are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that the interference will not occur in a particular installation, especially if you use a "rabbit ear" television antenna. (A "rabbit ear" antenna is the telescoping-rod type usually contained on TV receivers.)

You can determine whether your computer is causing interference by turning it off. If the interference stops, it was probably caused by the computer or its peripheral devices. To further isolate the problem:

- Disconnect the peripheral devices and their input/output cables one at a time. If the interference stops, it is caused by either the peripheral device or its I/O cable. These devices usually require shielded I/O cables. For Apple peripheral devices, you can obtain the proper shielded cable from your dealer. For non-Apple peripheral devices, contact the manufacturer or dealer for assistance.

If your computer does cause interference to radio or television reception, you can try to correct the interference by using one or more of the following measures:

- Turn the TV or radio antenna until the interference stops.
- Move the computer to one side or the other of the TV or radio.
- Move the computer farther away from the TV or radio.
- Plug the computer into an outlet that is on a different circuit than the TV or radio. (That is, make certain the computer and the radio or television set are on circuits controlled by different circuit breakers or fuses.)
- Consider installing a rooftop television antenna with coaxial cable lead-in between the antenna and TV.

If necessary, you should consult your dealer or an experienced radio/television technician for additional suggestions. You may find helpful the following booklet, prepared by the Federal Communications Commission:

*"How to Identify and Resolve Radio-TV Interference Problems."*

This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, stock number 004-000-00345-4.

# *Preface*

This is the reference manual for the Apple IIc personal computer. It contains detailed descriptions of all of the hardware and firmware that make up the Apple IIc. The manual is divided into two volumes: Volume I contains all the chapters; Volume II contains the appendixes.

This manual contains a lot of information about the way the Apple IIc works, but it doesn't tell you how to use the Apple IIc. For this, you should read the other Apple IIc manuals, especially the *Apple IIc Interactive Owner's Manual*.

This manual describes the internal operation of the Apple IIc as completely as possible in a single reference work. The criterion for deciding to include an item of information was whether it would help an assembly-language programmer or hardware designer.

---

## **Contents of This Manual**

This manual presents first the physical characteristics of the Apple IIc (Chapter 1), then the hardware locations and firmware that control memory management and input/output (Chapters 2 through 10), and finally the electrical and electronic implementation of those capabilities (Chapter 11). The appendixes contain summary tables and information comparing other Apple products to the Apple IIc.

Chapter 1 identifies the main physical features of the Apple IIc.

Chapter 2 presents an overview of the 65C02 microprocessor (whose instruction set appears in Appendix A), and then discusses the processor's address space, what it contains, and how to control it.

Chapter 3 is an introduction to Chapters 4 through 9. It describes the common characteristics of input/output processing. Chapters 4 through 9 then discuss the kinds of devices—both built-in and attachable—that the Apple IIc supports:

- Keyboard and speaker (Chapter 4)
- Video display (Chapter 5)
- Disk drives (Chapter 6)
- Serial port 1 for printers and plotters (Chapter 7)
- Serial port 2 for modems and other communication devices (Chapter 8)
- Mouse and hand controls, including game paddles and joysticks (Chapter 9)

Chapter 10 is a brief tutorial on how to use the Monitor firmware to disassemble and debug machine-language programs, and manipulate memory contents.

Chapter 11 is a detailed description of the hardware that implements the features described in the earlier chapters. This information is included primarily for programmers, but it will also help you if you just want to understand more about the way the Apple IIc works.

Additional reference information appears in the appendixes. Appendix A is the manufacturer's description of the 65C02 instruction set, including the 27 new instructions available on the CMOS version of the 65C02 used in the Apple IIc.

Appendix B is a memory map of the Apple IIc, including detailed tables of page zero, page three, the screen holes, and the hardware page.

Appendix C lists the *published* firmware entry points, arranged by address, and indicates where in the manual they are described. The list includes I/O firmware (pages \$C1 through \$CF) and Monitor firmware (pages \$F0 through \$FF). For Applesoft interpreter firmware (pages \$D0 through \$EF), refer to the *Applesoft BASIC Reference Manual* (in two volumes).

Appendix D discusses the operating systems and languages that run on the Apple IIc.

Appendix E describes how to use the Apple IIc's interrupt handling capabilities.

Appendix F contains an overview of the differences among the Apple II series computers.

Appendix G contains the keyboard layouts, code conversion tables, and external power supply characteristics of USA and international models of the Apple IIc.

Appendix H contains reference tables for code and number base conversion.

Appendix I contains a listing code for the Monitor, enhanced video firmware, and input/output firmware contained in the Apple IIc. The listings do not include the built-in Applesoft interpreter, which is discussed in the *Applesoft BASIC Programmer's Reference Manual*.

The bibliography lists articles and books containing additional information about the Apple IIc and related products.

The glossary defines many of the technical terms used in this manual.

At the back of this manual is a *Tell Apple Card*. Please fill it out and send it in. Your experience with this manual will help us plan new reference materials.

---

## ***Symbols in This Manual***

This manual uses a three-level numbering system to make it easier to cross-reference information. A reference like 2.4.3 means Chapter 2, section 4, subsection 3. G.1.8 refers to Appendix G, section 1, subsection 8.

Special text in this manual is set off in several different ways, as shown in these examples.



---

### **Warning**

*Important information regarding your safety or protection of data appears in boxes like this.*

---

Note: Information that is useful but not central to the discussion in a given part of the text appears in gray boxes like this.

Captions, cross-references and incidental definitions appear in marginal glosses like this.



# *Introduction*

This chapter introduces you to the Apple IIc. It identifies the major components of the machine, both outside and inside, and tells you where in the rest of the manual to find information about each one.

---

## 1.1 Outside of Machine

Appendix G illustrates and discusses several international keyboard layouts.

Figure 1-1 shows an Apple IIc with a Standard USA keyboard. This chapter discusses both the Standard (Sholes) and Simplified (Dvorak) USA keyboards, as well as the lights and switches on the front of the machine.

Figure 1-2 is a diagram of the parts of the computer that you can see, hear, or access from the outside. The Apple IIc comes equipped with keyboard, speaker (with headphone jack and volume control), disk drive, attachable power supply, and internal voltage converter. It also has built-in interfaces and connectors for a serial printer, video display, special video display adapters, modem, mouse, and game controls.

There are no user connections inside the Apple IIc, but expansion is possible and easy with the connectors on the Apple IIc back panel.

*Figure 1-1. Apple IIc With Standard USA Keyboard*



Keyboard  
(See Figs. 1-4 and 1-5)

Disk Drive  
(See Fig. 1-7)

*Figure 1-2. Block Diagram of External Features*



Back Panel  
(See Fig. 1-8)

Speaker  
(See Fig. 1-6)

**ASCII** stands for American Standard Code for Information Interchange. Table 4-2 lists the ASCII character encoding for the Standard and Simplified USA keyboards. Appendix G lists the encoding for international keyboards.

### **1.1.1 The Keyboard**

The front of the Apple IIc includes the keyboard (Figure 1-3), the computer's primary input device. It has a typewriter layout, uppercase and lowercase, and can generate all 128 characters—including control characters—in the ASCII character set. The front of the computer also has a reset key, 80/40 switch, keyboard switch, disk-use light, and power light.

Figure 1-3. Keyboard and Front of Apple IIc



Table 1-1 lists the characteristics of all Apple IIc keyboards and front panels.

**Table 1-1. Keyboard Specifications**

Number of keys:	63
Character encoding:	ASCII
Number of codes:	128
Features:	Automatic repeat, two-key rollover
Special function keys:	RESET, ( ), ( )
Cursor movement keys:	( ), ( ), ( ), ( ), RETURN, DELETE, TAB
Modifier keys:	CONTROL, SHIFT, CAPS-LOCK, ESC
Front-panel switches:	80/40 switch, keyboard switch
Front-panel lights:	Power light, disk-use light

### Features

The Apple IIc keyboard has automatic repeat on all character keys: if you hold the key down longer than about a second, the character it generates repeats. It also has two-key rollover, which means if you type a second key before releasing a prior key, the new character enters the computer the same as though the previous key were released first. (This is important for fast touch typists.)

The Apple keys are connected to one-bit addresses in memory, described in Chapter 9.



Chapter 2 describes the results of the various reset procedures.

The **Monitor** is a built-in program that performs some of the basic activities of the computer, such as retrieving and storing key codes as they come in, and clearing or updating the display screen.


The other keys to use with **CONTROL** are: @ [ \ ] ^ \_ and their international equivalents (see Appendix G).

---

### Special Function Keys

The Apple IIc has three special function keys: **RESET**, and two keys marked with apples: one outlined, , and one filled in, .

**RESET** has a direct line to the 65C02 microprocessor: holding down **CONTROL** while pressing **RESET** causes the Apple IIc to restart processing with a program that puts the machine in a known state (Chapter 2). So you don't accidentally destroy current work, the reset takes effect only if you hold down **CONTROL** while pressing **RESET**.

If you hold down both **CONTROL** and  while pressing **RESET**, the computer starts up as if you just turned it on.

---

### Cursor Movement Keys

Four of the cursor movement keys have arrows on them: left, right, down and up. The other three keys are **RETURN**, **DELETE** and **TAB**. All generate ASCII control characters (Table 4-2). However, it is up to the operating system or application program to interpret and act on the control codes that these keys generate.

---

### Modifier Keys

Three special keys, **CONTROL**, **SHIFT** and **CAPS-LOCK**, change the codes generated by the other keys. None of these keys generates a code when pressed by itself. A fourth key, **ESC**, generates a control code that the Monitor responds to by interpreting certain subsequent keystrokes in a modified way.

**CONTROL**, when pressed in combination with letter keys or certain other keys, produces ASCII control characters.

**SHIFT** works the same on the Apple IIc as on an ordinary typewriter: it selects uppercase letters and the upper characters on the keys.

**CAPS-LOCK**, in its down position, changes the letter keys to uppercase, but does not affect other keys.

**ESC** is not a modifier key in the same sense as **CONTROL** and **SHIFT**; you do not hold it down while pressing other keys. Rather, you press **ESC** and it generates the ESC control character (key code \$1B—see Table 4-2). Many programs, including the built-in Monitor program, then interpret other specific keys as designating an **escape sequence**.



---

### The 80/40 Switch

This switch takes effect only if the program or operating system you are using actually checks it. See Table 4-1.

The 80/40 switch selects whether information is to be displayed in 40 columns to the line or 80 columns. This switch indicates 40-column display in its down position, and 80-column display in its up position.

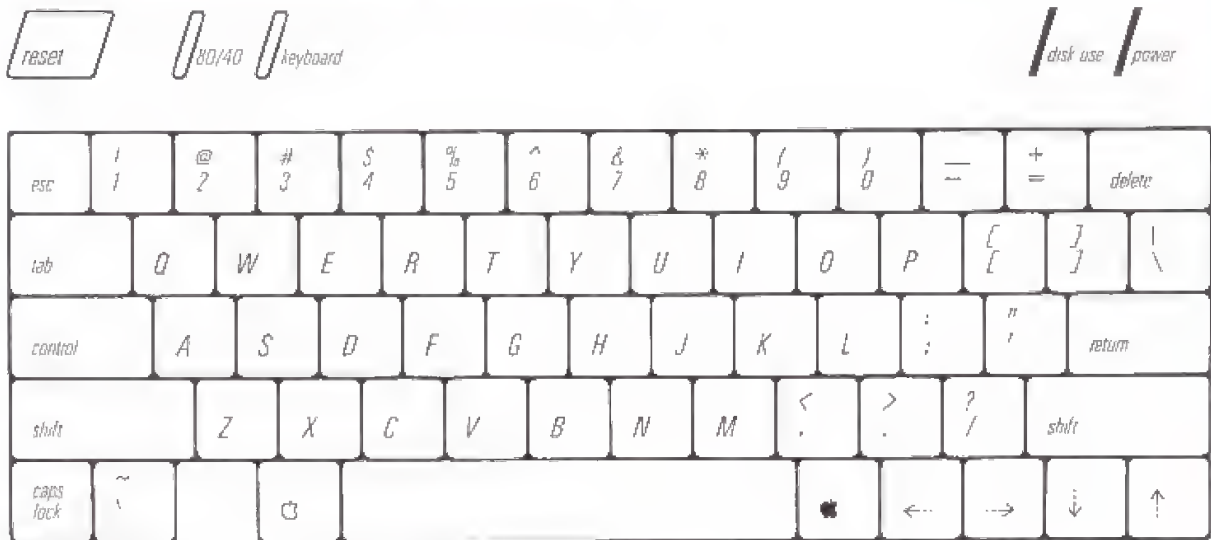
**Note:** Not all programs check this switch. Even programs that do check the switch may rely on the user setting it before running the program.

---

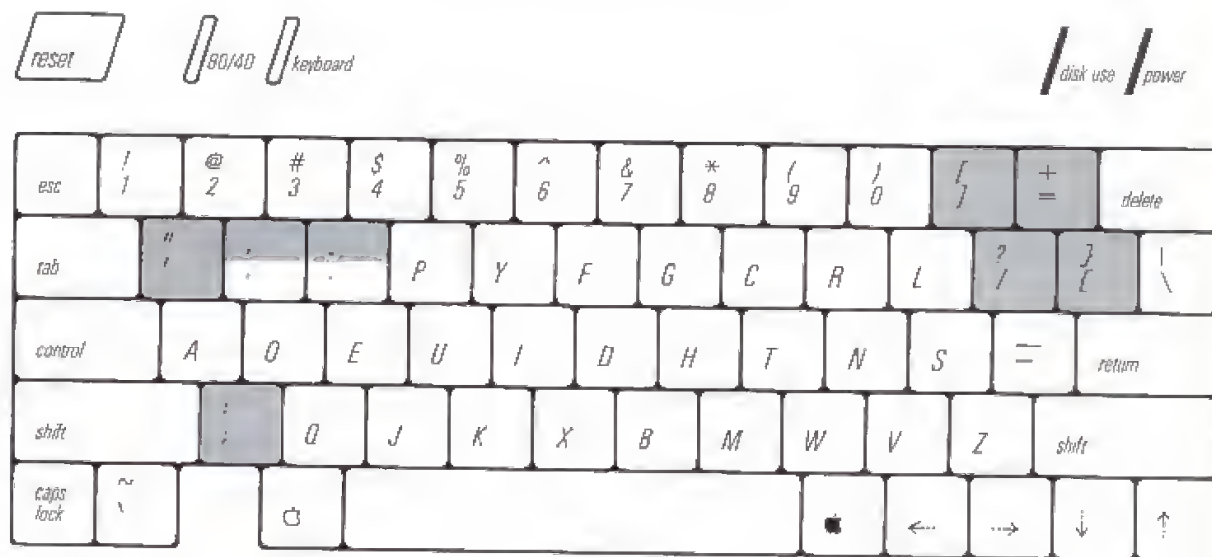
### The Keyboard Switch

The keyboard switch selects which of the two keyboard layouts and character sets the computer is to get from the keyboard and display on the screen. On USA versions of the Apple IIc, select the Standard Sholes keyboard layout (Figure 1-4) with the switch in the up position, and the Dvorak Simplified layout (Figure 1-5) with the switch in the down position.

Figure 1-4. The USA Standard or **Sholes** Keyboard (Keyboard Switch Up)



**Figure 1-5.** Simplified or **Dvorak** Keyboard (Keyboard Switch Down). Note: Shaded characters may be in different positions on some models.



Appendix G illustrates the keyboard layouts for both keyboard switch positions on several international versions of the Apple IIc.

On international models, the keycaps indicate the character positions for the local keyboard layout, which is selected when the keyboard switch is down. When up, the keyboard switch selects the USA Standard characters and key layout.

### **Disk-Use and Power Lights**

The red disk-use light glows whenever the built-in disk drive's motor is on.

The green power light glows when normal power is present at the Apple IIc's internal power supply.



### **Warning**

*If the power light flashes on and off, turn off the computer immediately. Find out what caused the condition (such as a brownout) and remedy it before turning the computer on again. Above all, do not use the disk drive when the power light is flashing; this may damage the computer.*

### 1.1.2 The Speaker

The Apple IIc has a loudspeaker in the bottom of the case, as shown in Figure 1-6. The speaker enables Apple IIc programs to produce a variety of sounds that make programs more useful and interesting. There is also a volume control on the left side of the Apple IIc case, and a mini-phone jack for connecting headphones or an external speaker.

The way programs control the speaker is described in section 4.2.

The jack accepts either one-channel (monaural) or two-channel (stereo) plugs, although speaker output is monaural only. Inserting a plug disconnects the built-in speaker.

*Figure 1-6. Speaker, Volume Control, and Phone Jack*

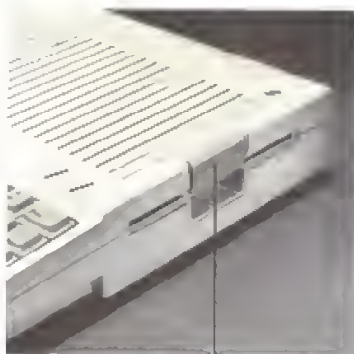


I/O means input (information coming into the computer) and output (information going out of the computer). Chapter 6 describes how to use the Apple IIc's disk I/O hardware and firmware.

### 1.1.3 The Built-in Disk Drive

The Apple IIc has a built-in disk drive (Figure 1-7) that is fully compatible with Apple Disk II—that is, it reads and writes single-sided, 35-track disks. The drive door is on the right side of the Apple IIc case.

Figure 1-7. Built-in Disk Drive



Disk Drive Door

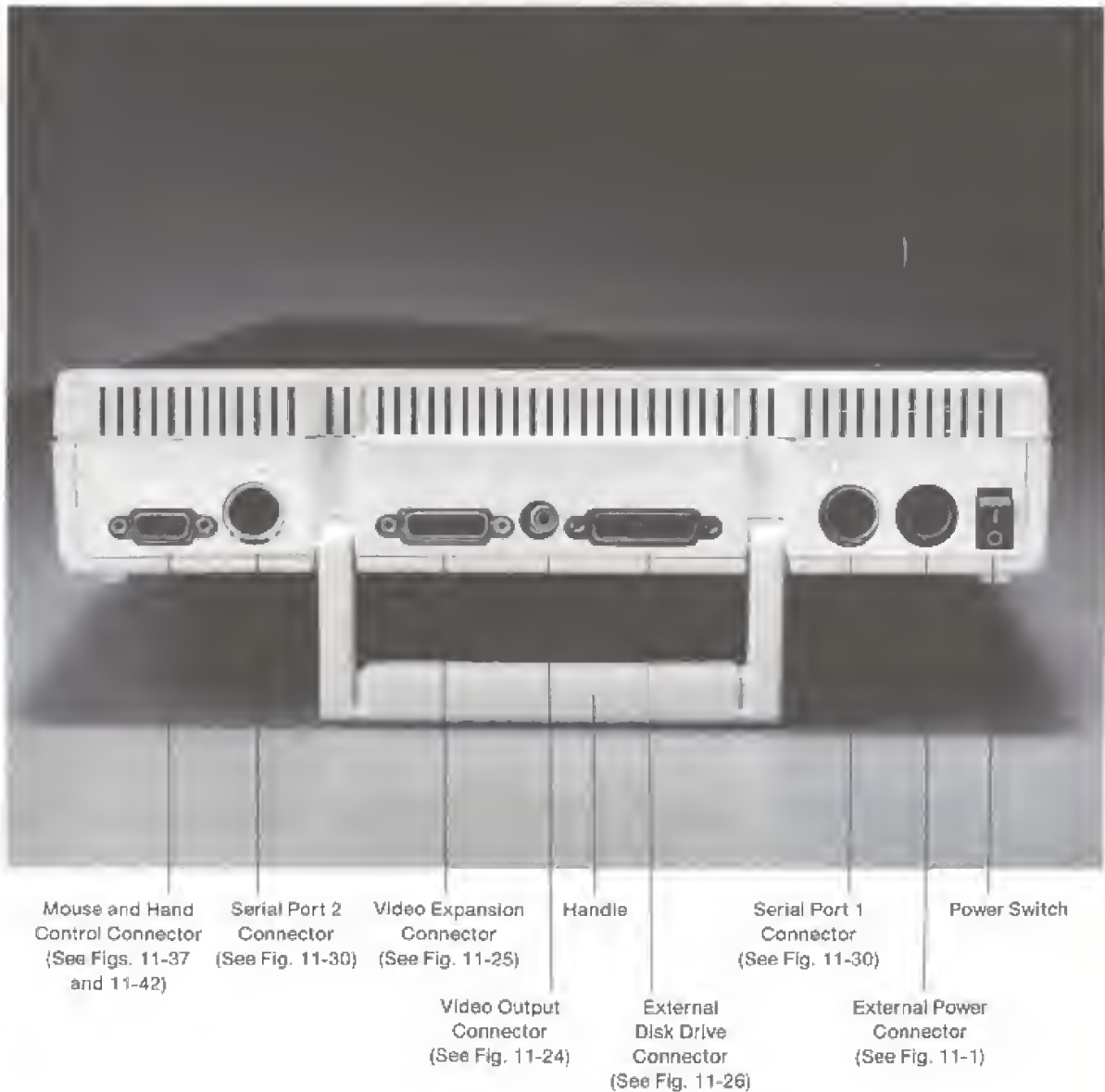
### 1.1.4 The Back Panel

The back panel of the Apple IIc (Figure 1-8) has seven connectors and a main power switch. From left to right they are:

- a 9-pin D-type miniature connector for connecting hand controls, a mouse, a joystick or some other pointing device
- a 5-pin DIN connector for serial input and output (port 2; normally for a modem)
- a 15-pin D-type connector for video expansion
- an RCA-type jack for a video monitor
- a 19-pin D-type connector for connecting a second disk drive
- another 5-pin DIN connector for serial input and output (port 1; normally for a printer or plotter)
- a special 7-pin DIN connector for power input

The installation manuals for the external devices contain instructions for connecting them. Be sure to move the handle until it clicks into position for propping up the computer before attaching cables to the back panel.

*Figure 1-8. Back Panel Connectors*



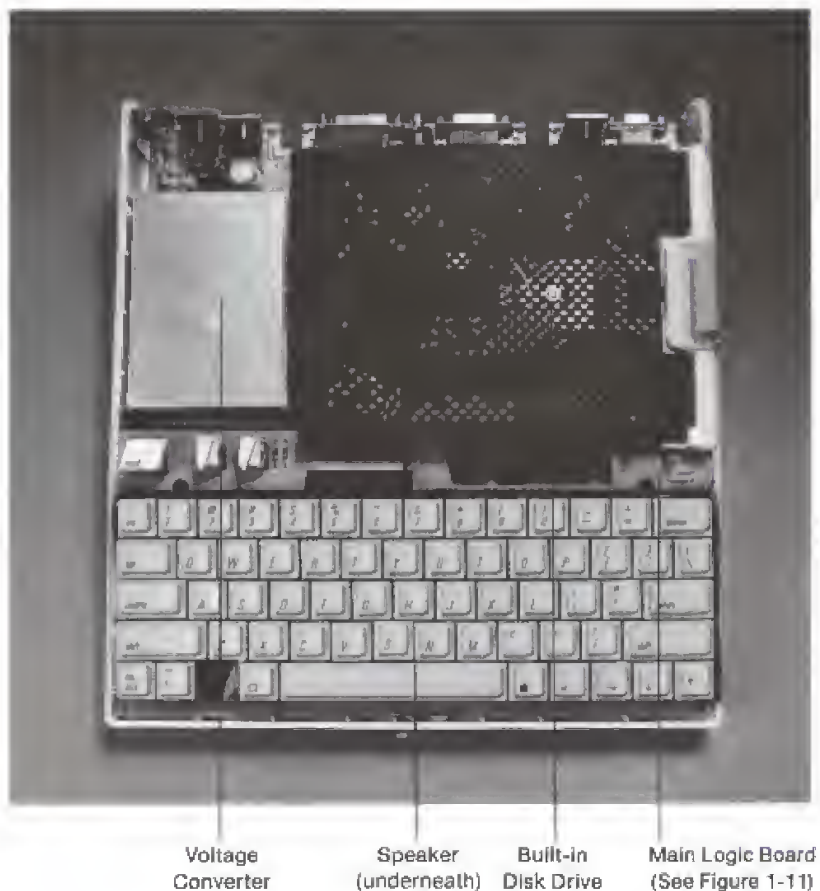


## 1.2 Inside of Machine

Chapter 11 discusses in further detail these components and how they work.

Figure 1-9 shows the main components inside the Apple IIc computer.

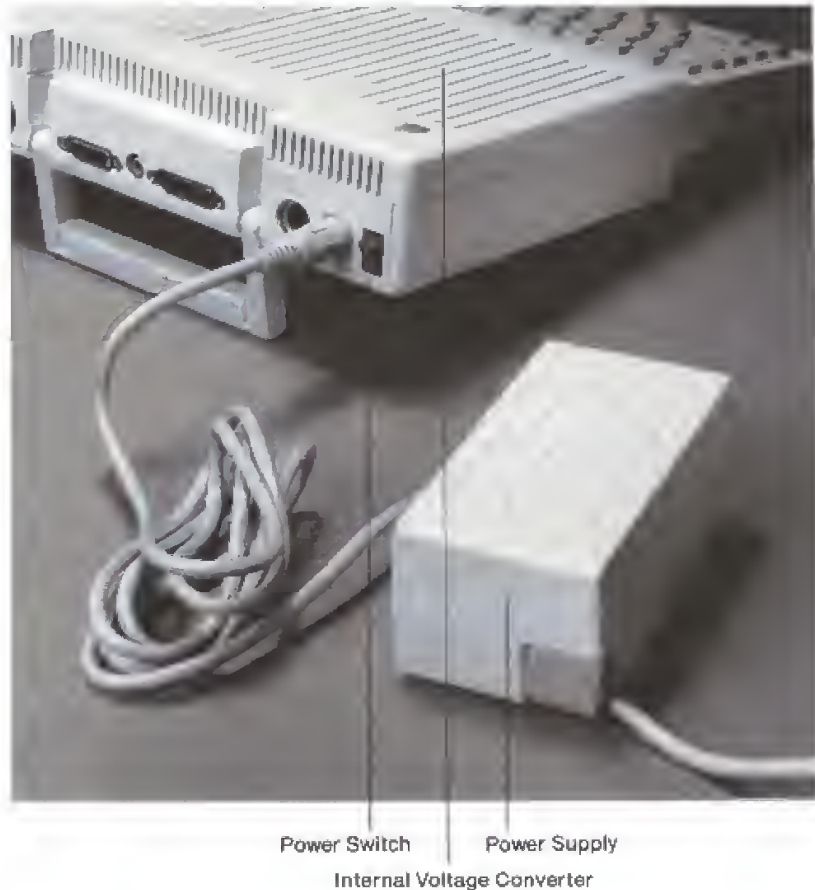
*Figure 1-9. Block Diagram of Inside of Machine*



### 1.2.1 The Internal Voltage Converter

The built-in voltage converter operates from a 15V DC source, such as provided by the external power supply furnished with the Apple IIc (Figure 1-10). The voltage converter provides power for the logic board, built-in disk drive, one external disk drive, and the I/O signals available at the back panel.

*Figure 1-10. Power Supply and Voltage Converter*



Complete specifications of the Apple IIc power supply and voltage converter appear in Chapter 11.

The voltage converter produces three different voltages: +5V, +12V, and -12V. (Minus 5V is derived from -12V on the main logic board.) It is a high-efficiency switching converter that protects itself and the rest of the Apple IIc against short circuits and other mishaps.

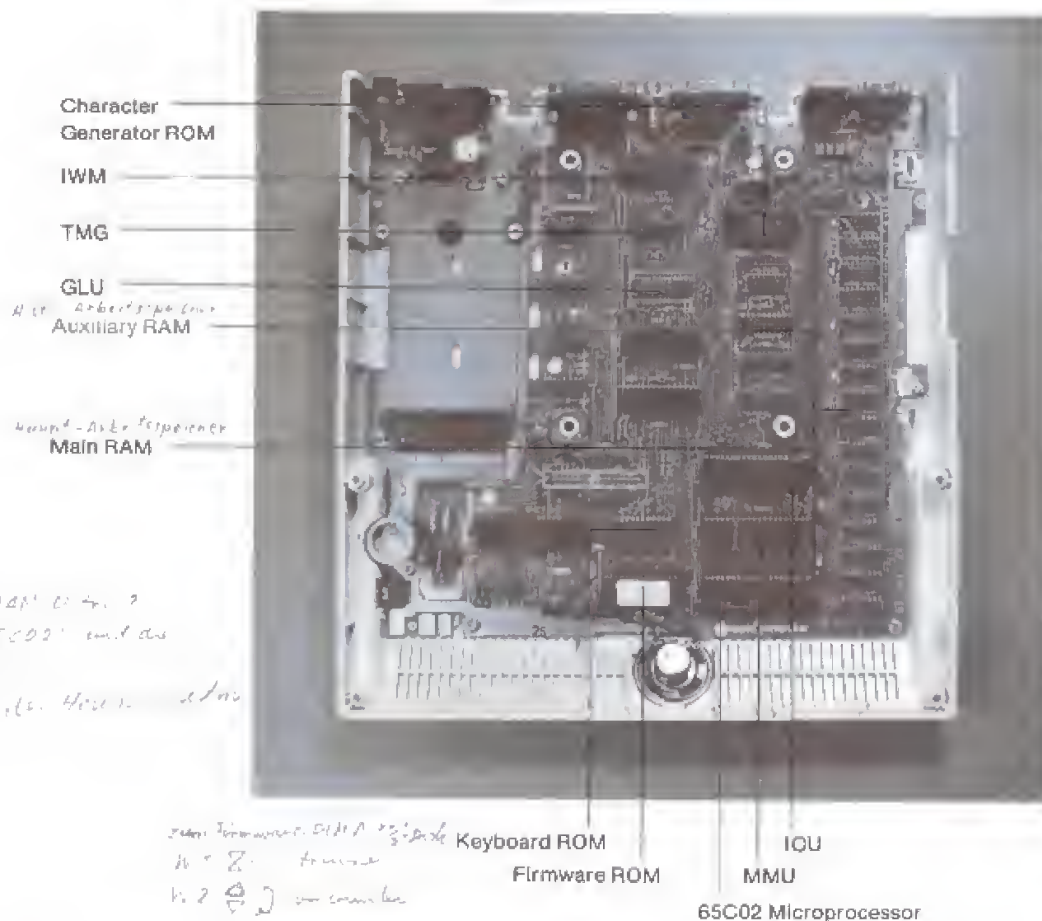
### **1.2.2 The Main Logic Board**

Almost all the electronic parts of the Apple IIc are attached to the main logic board, which is mounted flat in the bottom of the case.

**Firmware** is program code that is stored in read-only memory. It can be read and executed, but not changed.

Figure 1-11 shows the main logic board and the most important integrated circuits (ICs) in the Apple IIc. They are the CPU (central processing unit), RAM (random access memory), ROM (read-only memory) ICs for keyboard encoding, display character generation, and firmware, and the five custom integrated circuits.

Figure 1-11. Main Logic Board



The specifications of the 65C02 are given in Chapter 11; the 65C02 instruction set is given in Appendix A.

The CPU is a 65C02 microprocessor. The 65C02 is a CMOS version of the 6502, which is an eight-bit microprocessor with a sixteen-bit address bus. In the Apple IIc, the 65C02 runs at 1 MHz and performs up to 500,000 eight-bit operations per second.

Chapter 11 describes how RAM works; Appendix B lists important RAM locations.

ROMs: see Chapter 11.

The Applesoft language interpreter is described in the *Applesoft Tutorial* and the *Applesoft BASIC Programmer's Reference Manual*.

Memory addressing: see Chapter 2.

See Chapters 3 through 9.

Chapter 11 discusses the functions of these integrated circuits in some detail.

The keyboard is scanned by an IC that generates matrix values for a read-only memory (ROM). The value of this ASCII code is latched and readable by programs.

The character generator ROM converts display information to a form that display devices can use.

The other ROM contains the Monitor, the Applesoft BASIC interpreter, enhanced video firmware, and other input/output firmware. The firmware that this ROM contains is described throughout this manual.

Five of the large IC's are custom-made for the Apple IIc:

- The Memory Management Unit (MMU) contains most of the logic that controls memory addressing in the Apple IIc.
- The Input/Output Unit (IOU) contains most of the logic that controls the built-in input and output features of the Apple IIc.
- The Timing Generator (TMG) generates all the system and I/O clock and timing signals from a 14 MHz oscillator.
- The General Logic Unit (GLU) performs the remaining logic functions required.
- The Integrated Woz Machine (IWM) is a single-chip version of the Apple Disk II controller card.

---

### 1.2.3 The Other Circuit Boards

The Apple IIc contains other circuit boards that serve special purposes: a motor-speed control board and a read/write logic board for the disk drive, and a matrix board for detecting the position of keys pressed. This manual does not discuss these circuit boards.



---

#### Warning

*Adjustment of disk drive speed must be done by an authorized Apple Service Center. Do not attempt to adjust the speed of your built-in disk drive. If you do, you may damage it and you will void your warranty.*

---

# *Memory Organization and Control*

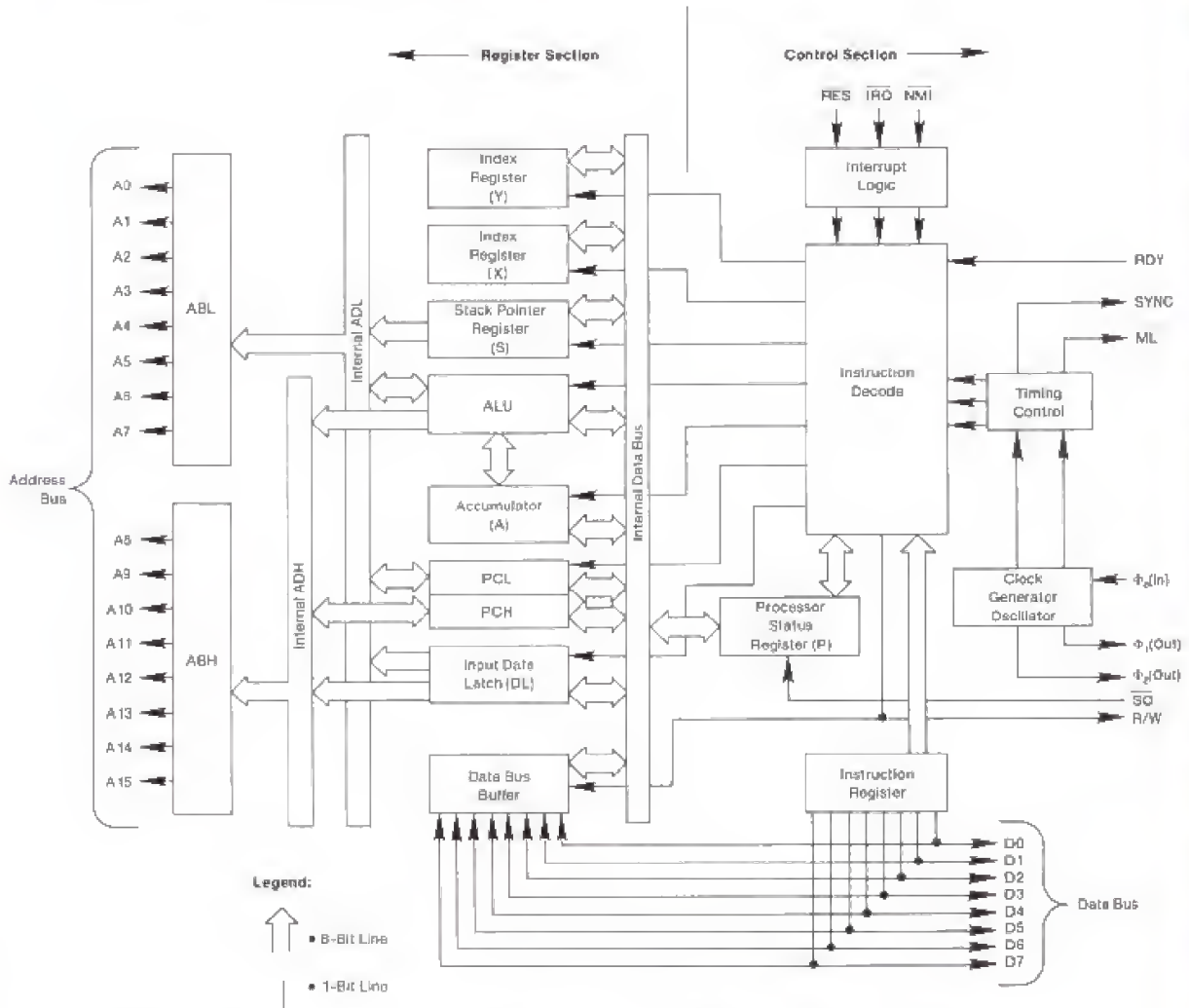


This chapter is an introduction to the microprocessor, the number of separate locations (addresses) it can access, and the addresses set aside for special purposes. The last section of this chapter describes the reset routines, which restore the computer to a known state.

## 2.1 The 65C02 Microprocessor

Figure 2-1 is a model of the 65C02 microprocessor. The 65C02 has one 16-bit register and five 8-bit registers. **Registers** are fast-acting storage areas where the processor performs and keeps track of its work.

**Figure 2-1. Block Diagram Model of 65C02.** Copyright 1982, NCR Corporation.  
Used by permission of NCR Corporation, Dayton, Ohio.



Each of the other registers holds eight bits (one byte), so the 65C02 is called an **8-bit processor**.

The 16-bit register is called the **program counter (PC)**. It specifies the address in memory that contains the instruction the processor is currently carrying out. A sixteen-bit register can specify any one of 65,536 memory addresses, and so the 65C02 is said to have an address space of 65,536 locations.

Appendix A lists the instructions the 65C02 can carry out, their use, and their effects on the registers. For further information, consult the pertinent books listed in the Bibliography.

The five 8-bit registers in the 65C02 are

- The **accumulator**, or A register. The accumulator is like a desktop where the processor performs mathematical and logical operations on information.
- The **index registers**, X and Y. The processor uses these registers to modify the address where information is to be found or placed, and to pass information from one program to another.
- A **stack pointer**, or S register. The processor uses a 256-byte region of memory—page 1—as an area to stack up bytes for future use. The stack is empty when the computer is turned on. Several 65C02 instructions either *push* (store) the contents of a register onto the stack, or *pull* (retrieve) a byte from the stack and place it in a register. The S register keeps track of the byte in the stack that is currently ready for use.
- A **processor status register**, called the P register. Seven of the eight bits of this register store flags that record the outcome of processor activities, and that can be checked by later instructions to determine what the processor should do next.

---

## 2.2 Overview of the Address Space

Soft switches are described in sections 2.4 and 2.5.

The Apple IIc's 65C02 microprocessor can address 65,536 (64K) memory locations. All of the Apple IIc's RAM, ROM, and input and output (I/O) devices are accessed using addresses in this 64K address range. Some functions have the same addresses—but not at the same time. The Apple IIc controls its shared addresses using soft switches.

**Note:** When referring to memory space, K stands for 1024, which is 2 to the tenth power. It is called *K* because 1024 is very close to the value 1000, which has long been abbreviated K for *Kilo*. Some early computers even saved the extra 24 locations for spares.

**RAM** stands for random-access (readable and writable) memory. **ROM** means read-only memory. Refer to the Glossary for further information.

There are two other ROMs in the Apple IIc: one to generate characters corresponding to keystrokes (section 11.7), and another to generate characters for display (section 11.9). However, these ROMs are not addressable by the microprocessor.

All input and output in the Apple IIc is memory mapped—that is, specific memory addresses (all in the \$C0 page) are allocated to each I/O device. In this chapter, the I/O memory spaces are described simply as areas of memory. For details of the built-in I/O features and firmware, refer to the descriptions in Chapters 3 through 9.

A block of 256 address locations is called a **page**. A one-byte address counter or 8-bit register can specify one of 256 different locations. Thus, page 0 consists of memory locations from 0 to 255 (hexadecimal \$0 to \$FF), inclusive; page 1 consists of locations 256 to 511 (hexadecimal \$100 to \$1FF); and so on. In this manual, all page numbers (except some of the low-numbered ones) are given in hexadecimal format.

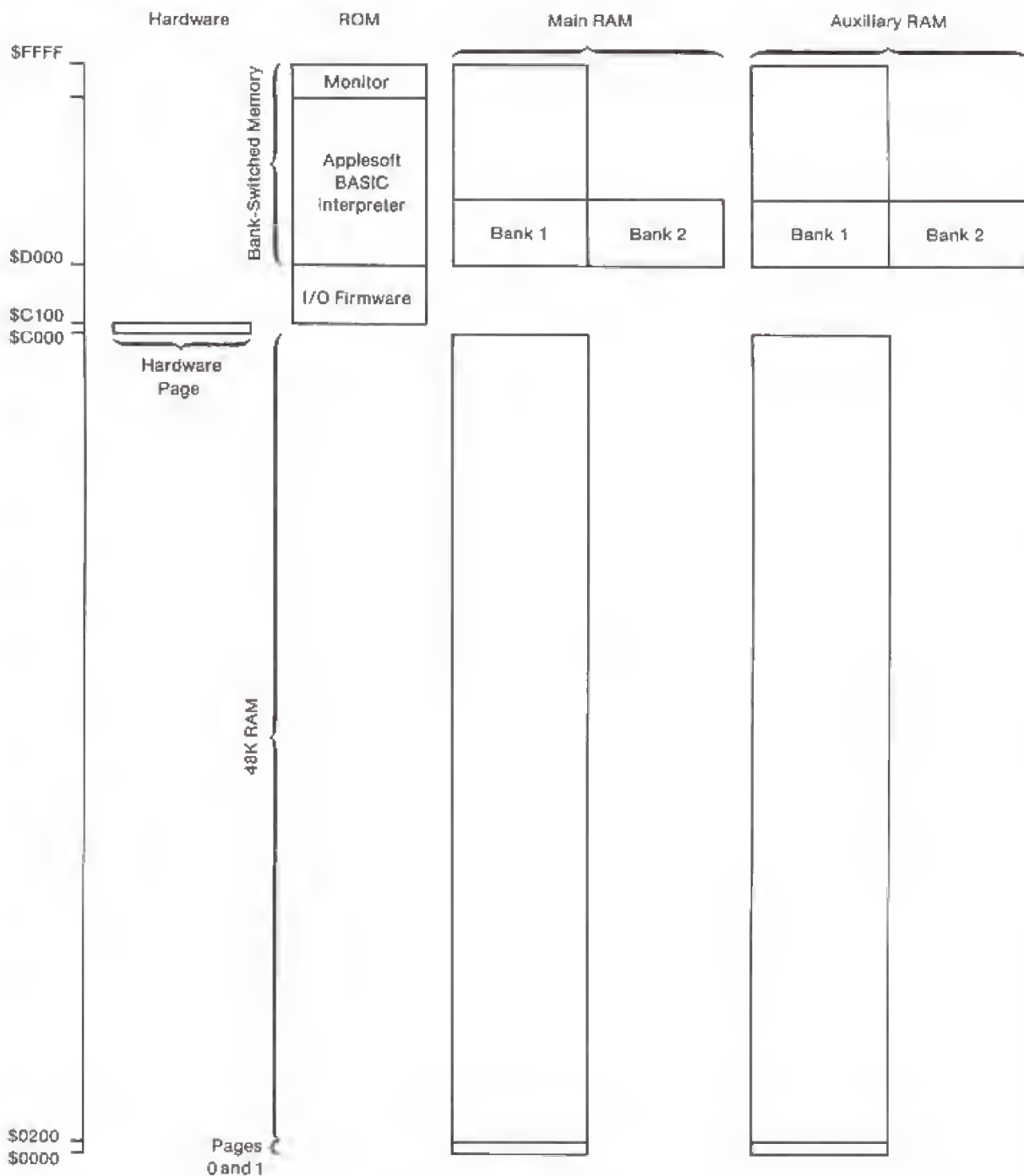
**Note:** The first two digits of a four-digit hexadecimal address are the page number. There are 256 pages of 256 bytes each in the address space. This kind of page is different from the display areas in the Apple IIc, which are sometimes referred to as Page 1 and Page 2.

---

## 2.3 Memory Map and Memory Switching

Figure 2-2 is a map of the Apple IIc's memory address space and what the major blocks of addresses are used for. As you can see in the figure, addresses \$C000 through \$C0FF contain hardware only, and addresses \$C100 through \$CFFF contain ROM only. At all other addresses there are two, three or even five blocks of RAM or ROM locations. At any given time, only one block of RAM or ROM occupies each set of addresses. As described later in this chapter, switches in the hardware page control which blocks the processor is to use.

Figure 2-2. Apple IIc Memory Map





---

### **2.3.1 Main RAM Addresses (\$0000-\$BFFF and \$D000-\$FFFF)**

The area labeled *Main RAM* in Figure 2-2 is so-called because some or all of it is present in all models of the Apple II series of computers. The Apple IIc has 64K bytes of main RAM.

---

### **2.3.2 Auxiliary RAM Addresses (\$0000-\$BFFF and \$D000-\$FFFF)**

The Apple IIc has another 64K of auxiliary RAM built in. Some or all of auxiliary memory is present in an Apple IIe with one of the 80-column cards installed (Appendix F), but there is no auxiliary RAM in the Apple II or II Plus. This portion of RAM cannot be used simultaneously with main RAM; you must use the soft switches described in this chapter to select main or auxiliary memory for a given range of addresses.

---

### **2.3.3 ROM Addresses (\$C100-\$FFFF)**

ROM addresses contain the built-in Apple IIc firmware. Addresses \$C100 through \$CFFF belong exclusively to ROM. Addresses \$D000 through \$FFFF are shared by ROM, main RAM, and auxiliary RAM; the selection techniques are described in section 2.4.2.

Pages \$C1 through CF (addresses \$C100 through \$CFFF) contain I/O firmware. The following associations apply for the Apple IIc:

- Serial port 1 (RS-232 device) firmware entry points are on page \$C1.
- Serial port 2 (communication device) firmware entry points are on page \$C2.
- Video output firmware entry points are on page \$C3; the enhanced video firmware and miscellaneous I/O support routines occupy pages \$C8 through \$CF.
- Mouse firmware entry points are on page \$C4.
- Disk I/O firmware entry points are on page \$C6.

**Note:** This correspondence of ports and entry points does not imply that all of each port's firmware occupies a specific page. The Apple IIc I/O port firmware space is allocated in a way that provides the best possible performance.

The operation of the Applesoft Interpreter firmware is described in the *Applesoft BASIC Programmer's Reference Manual*.

Pages \$D0 through \$F7 (addresses \$D000 through \$F7FF) contain the Applesoft Interpreter firmware.

Pages \$F8 through \$FF (addresses \$F800 through \$FFFF) contain the Monitor, which is described in Chapter 10. Monitor routines that make various input and output procedures easier are described in Chapters 3 through 9.

Chapters 3 through 9 describe the Apple IIc's input and output locations. Appendix B lists all of these locations in address order, rather than by function.

---

### 2.3.4 Hardware Addresses (\$C000-\$C0FF)

The Apple IIc's built-in input and output functions, and the switching of blocks of address space, all take place via locations on the \$C0 page—that is, in the address range \$C000 through \$C0FF. This chapter describes the address space (memory) switches.

Bit numbering in a byte is explained in Appendix H.

The hardware functions on this page fall into five basic categories:

- **Data inputs.** The only data input is location \$C000, where the low-order seven bits (bits 6 through 0) represent the keyboard key just pressed. (This data is guaranteed valid only when bit 7 = 1.)
- **Flag inputs.** Most built-in input locations are single-bit flags in the high-order (bit 7) position of their respective memory addresses. Flags have only two values: on (greater than or equal to 128 or \$80) or off (less than 128 or \$80).

The switch, hand controller (analog) and button inputs, and the keyboard strobe, are examples of flag inputs. The locations for reading soft-switch states are also of this type.

- **Strobe outputs.** The clear keyboard strobe (Chapter 4) and paddle timer strobe (Chapter 9) outputs are controlled by memory locations. If your program reads the contents of one of these locations, then the function associated with that location will be activated.

- **Toggle switches.** The Apple IIc has only one toggle switch: the speaker switch. A toggle switch has only one address assigned to it; each time you access it, it changes to its other state (on or off).

Reading the speaker toggle at location \$C030 clicks the speaker once. However, if you write to the speaker location, the microprocessor activates the address bus twice during successive clock cycles, causing the speaker toggle to end up in its original state before the speaker cone can move. Therefore, you should read, rather than write, to use this device.

The processor cannot read the on/off status of the speaker switch.

- **Soft switches.** Soft switches are two-position switches turned on by accessing one address and turned off by accessing another address. Most of these switches have a third address associated with them for reading the state of the switch.

There are eight soft switches that select different combinations of bank-switched memory (section 2.4). Four of these eight switches require that your program read them twice in succession to activate them.

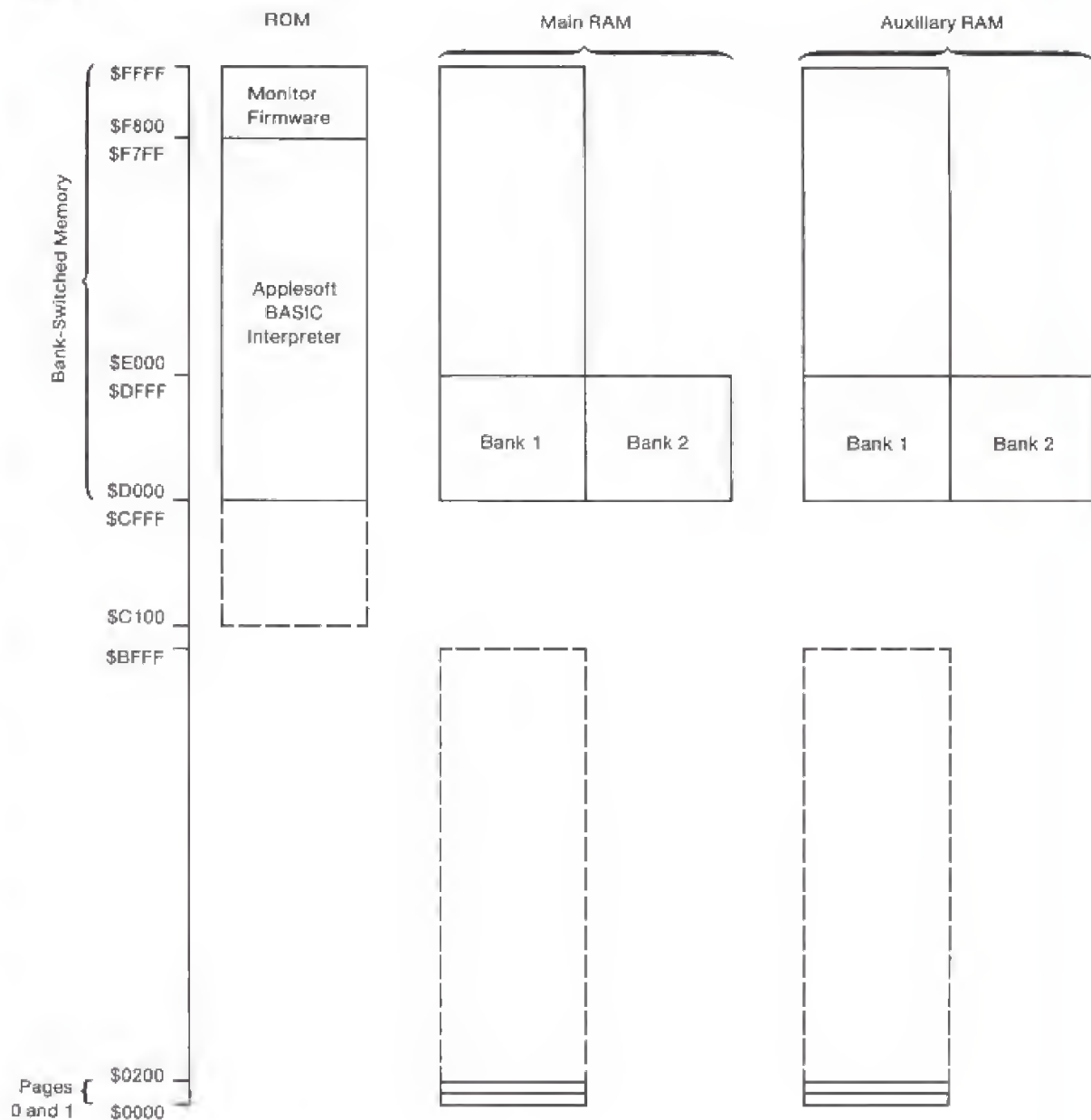
## 2.4 Bank-Switched Memory

The stack and zero page are switched this way so that system software running in the bank-switched memory space can maintain its own stack and zero page while it manipulates the 48K memory space (section 2.5).

Section 2.4.1 discusses what functions various addresses are set aside for; section 2.4.2 describes how to select the memory banks you want.

The memory areas described in this section are called **bank-switched memory** (Figure 2-3) because so many banks (ranges) of addresses—one bank of ROM and up to four banks of RAM—occupy the same group of locations among the upper addresses of memory. Pages \$00 and \$01, at the low end of memory, are included here because the two sets of them—one in main RAM and one in auxiliary RAM—are controlled by the same switches as the high-address banks.

Figure 2-3. Bank-Switched Memory



---

### ***2.4.1 Page Allocations***

Pages zero and one are used by many of the 65C02 instructions. The ROM and RAM addresses in bank-switched memory are usually occupied by system software such as interpreters, compilers, and operating systems.

---

#### ***Page \$00 (One-Byte Addresses)***

Several of the 65C02 microprocessor's addressing modes—for example, indirect addressing—require the use of addresses in page zero, or zero page. However, the Monitor, the interpreters, and the operating systems all make extensive use of page zero, too. One way to avoid conflicts is to use only those page-zero locations not already used by these other programs. But there is another way.

See Table B-1 in Appendix B.

As you can see from Table B-1, page zero is pretty well used up, except for a few bytes here and there. Rather than trying to squeeze your data into an unused corner, you may prefer a safer alternative: turn off interrupts, save the contents of part of page zero, use that part, then restore the previous contents to page zero, restore interrupts to their previous state, and then pass control to another program.

---

#### ***Page \$01 (The 65C02 Stack)***

The 65C02 microprocessor uses page 1 as its stack—a place where it can store subroutine return addresses, in last-in, first-out sequence. Programs can also use the stack for temporary storage of registers (via push and pull instructions). However, programs should use the stack carefully.

---

#### ***Pages \$D0 Through \$FF (ROM and RAM)***

All these memory banks are controlled by the soft switches described in section 2.4.2.

The memory address space from \$D000 through \$FFFF is used for both ROM and RAM. The 12K bytes of ROM in this address space contain the Monitor and the Applesoft BASIC interpreter.

There are 16K bytes of main RAM in this 12K space, with two banks occupying the 4K of addresses from \$D000 through \$DFFF. The RAM is normally used for storing other languages such as Pascal, or operating systems such as ProDOS.

There are also 16K bytes of auxiliary RAM in this 12K space, again with double occupancy in the address range \$D000 through \$DFFF.



---

### 2.4.2 Using Bank Selector Switches

You switch banks of memory in the same way you switch other functions in the Apple IIc: by using soft switches. These soft switches do four things:

1. Select either RAM or ROM in this memory space.
2. Allow or inhibit (write-protect) writing to the RAM when RAM is selected.
3. Select the first or second 4K-byte bank of RAM in the address space \$D000 to \$DFFF.
4. Select either main RAM or auxiliary RAM.



---

#### Warning

*Do not use soft switches without careful planning. Careless switching between RAM and ROM is almost certain to have catastrophic effects on your program.*

---

Table 2-1 shows the addresses of the soft switches for selecting all allowed combinations of reading and writing in this memory space, and the addresses of the locations to read the switch settings. Figures 2-4 through 2-10 illustrate how to select the combinations and what the resulting status of each switch is.

To make sure you do not inadvertently remove write protection from bank-switched RAM, the four write-enable addresses require that you read them twice in succession (indicated by RR in Table 2-1).

Because the ALTZP switch shares the read keyboard address, you must write (W in Table 2-1) to its locations to change the switch setting.

To find out which way a switch is set, read the appropriate location and then check bit 7 (shown as R7 in Table 2-1). If the bit is a 1, the answer to the question given in the table is affirmative.

**Note:** There is no way to check whether write protection is on or off.

*Table 2-1. Bank Select Switches*

Action	Hex	Name	Function
R	\$C080		Read RAM; no write; use \$D000 bank 2.
RR	\$C081		Read ROM; write RAM; use \$D000 bank 2.
R	\$C082		Read ROM; no write; use \$D000 bank 2.
RR	\$C083		Read and write RAM; use \$D000 bank 2.
R	\$C088		Read RAM; no write; use \$D000 bank 1.
RR	\$C089		Read ROM; write RAM; use \$D000 bank 1.
R	\$C08A		Read ROM; no write; use \$D000 bank 1.
RR	\$C08B		Read and write RAM; use \$D000 bank 1.
R7	\$C011	RDBNK2	Read whether \$D000 bank 2 (1) or bank 1 (0).
R7	\$C012	RDLCRAM	Reading RAM (1) or ROM (0).
W	\$C008	ALTZP	Off: use main bank, page 0 and page 1.
W	\$C009	ALTZP	On: use auxiliary bank, page 0 and page 1.
R7	\$C016	RDALTZP	Read whether auxiliary (1) or main (0) bank.

Figure 2-4. Read ROM

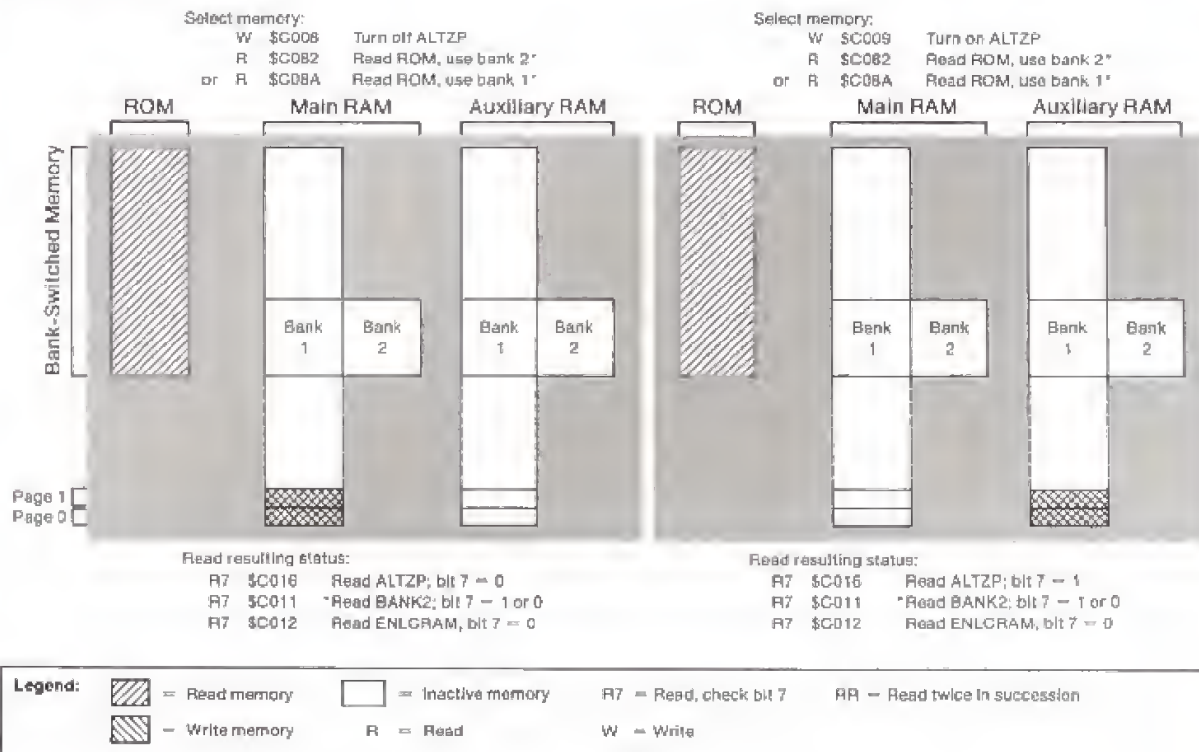


Figure 2-5. Read ROM, Write RAM, and Use First \$D0 Bank

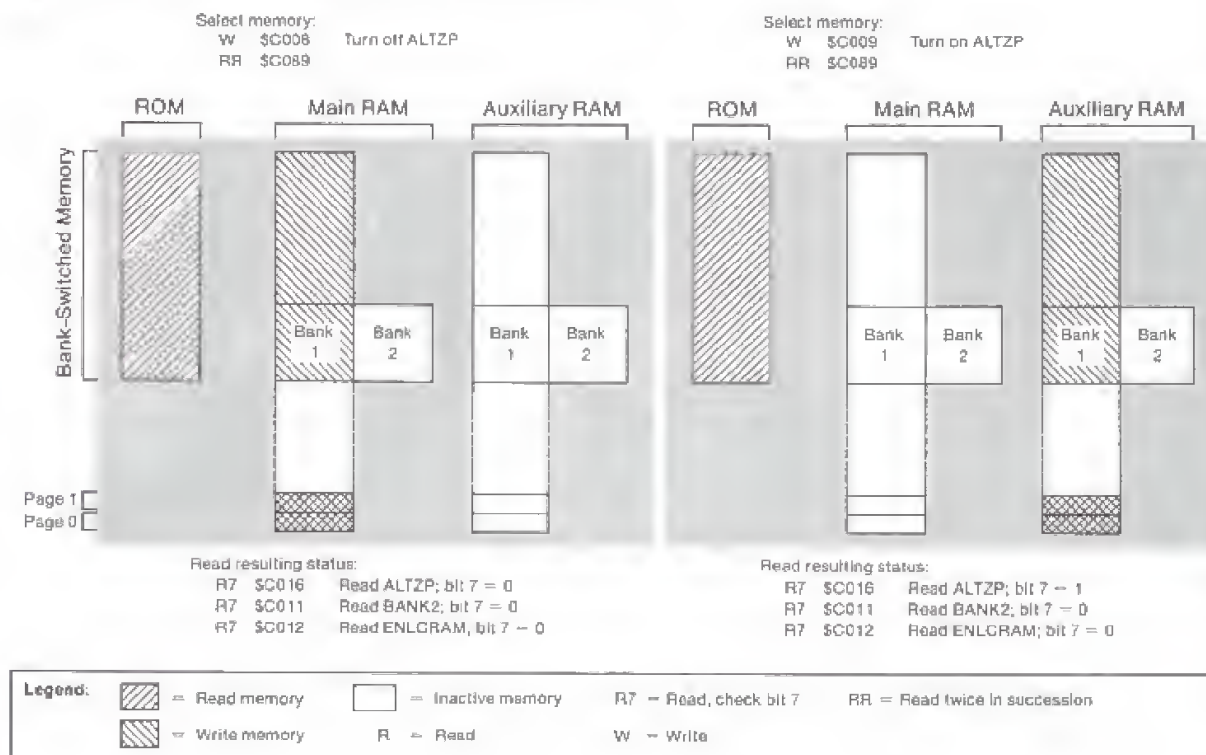


Figure 2-6. Read ROM, Write RAM, and Use Second \$D0 Bank

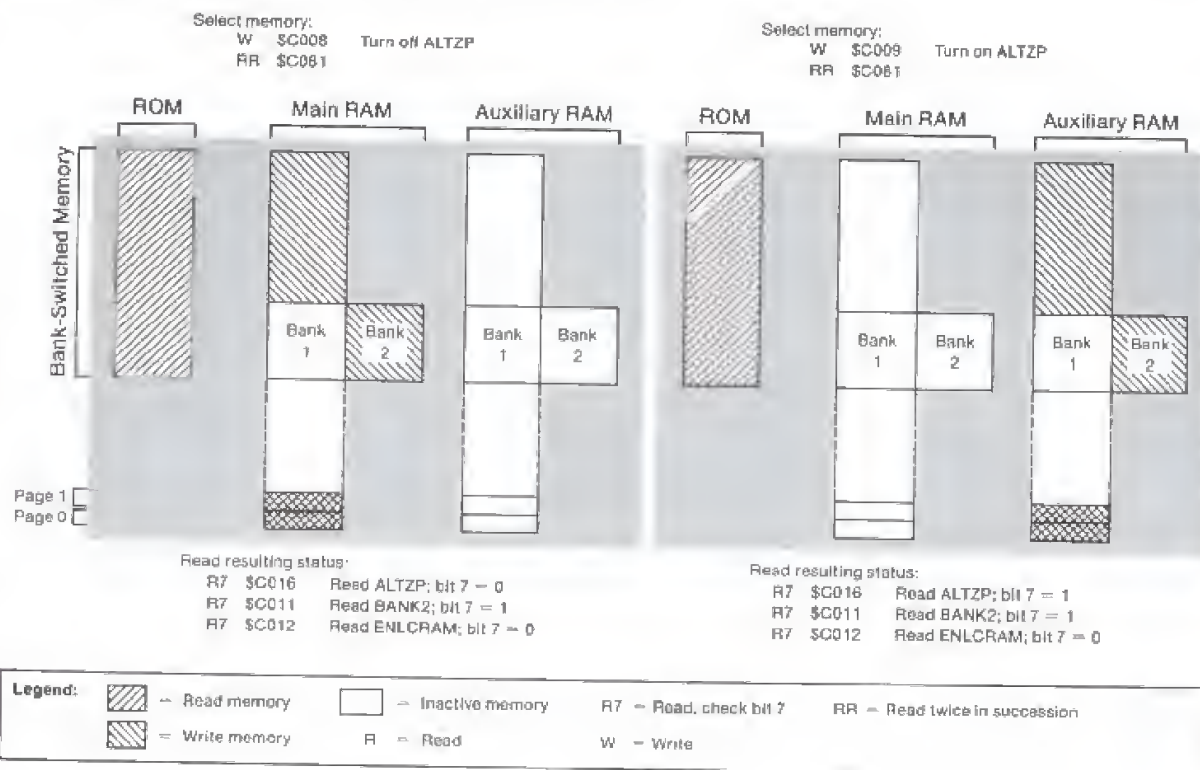




Figure 2-7. Read RAM and Use First \$D0 Bank

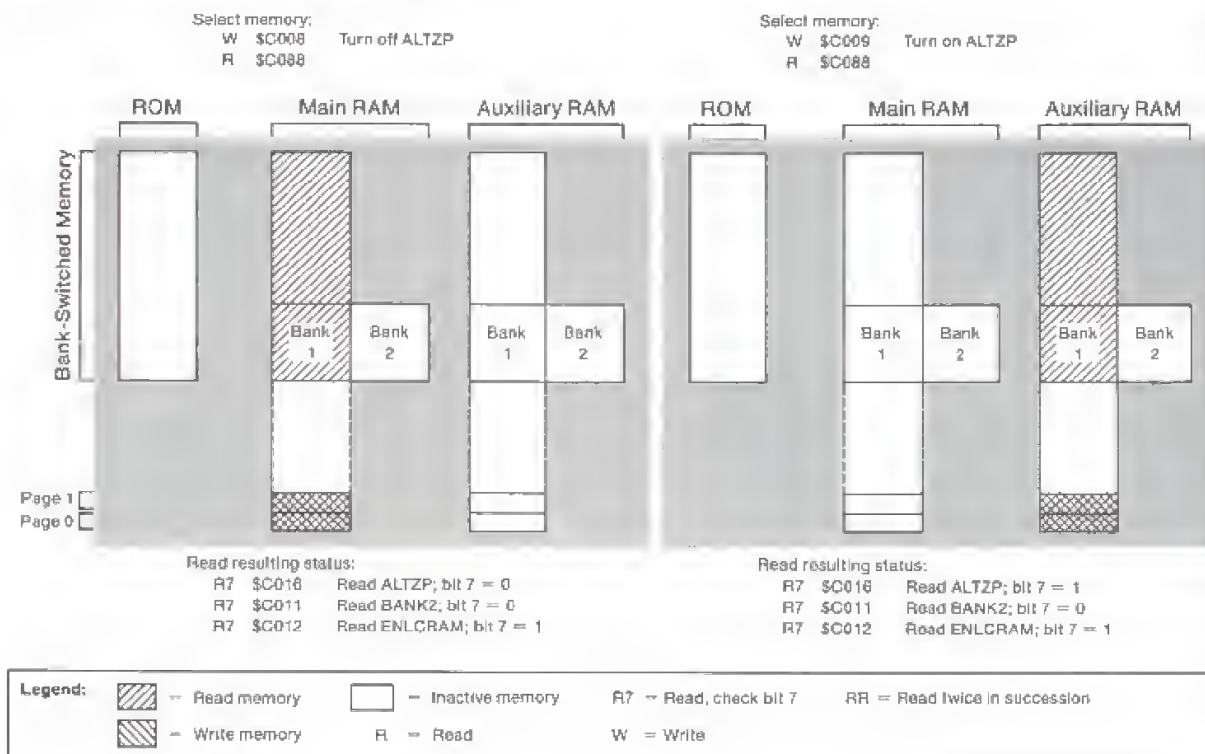
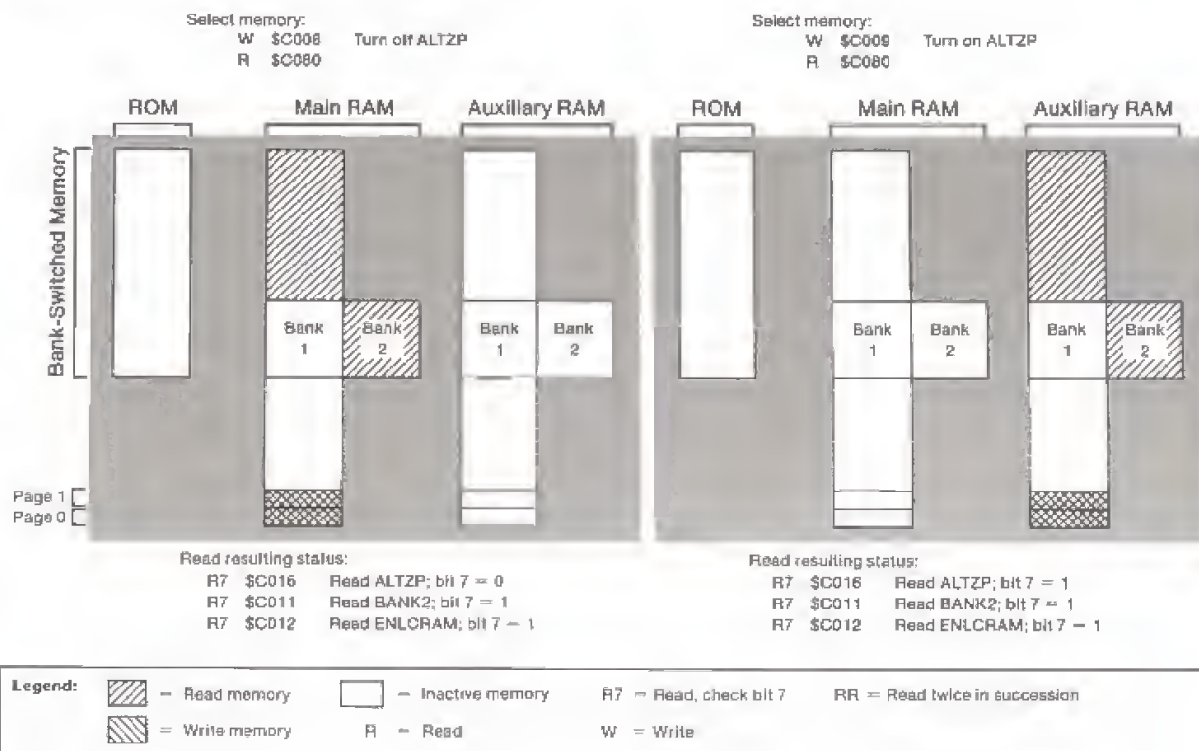


Figure 2-8. Read RAM and Use Second \$D0 Bank



**Figure 2-9. Read and Write RAM and Use First \$D0 Bank**

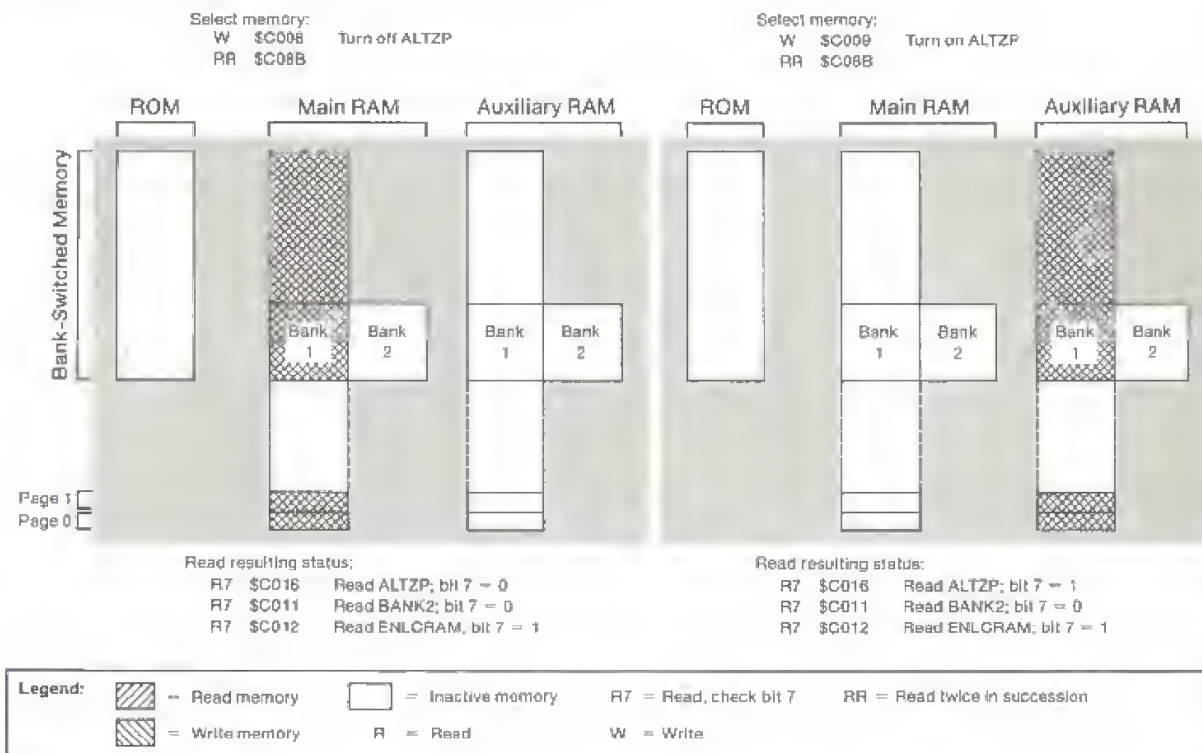
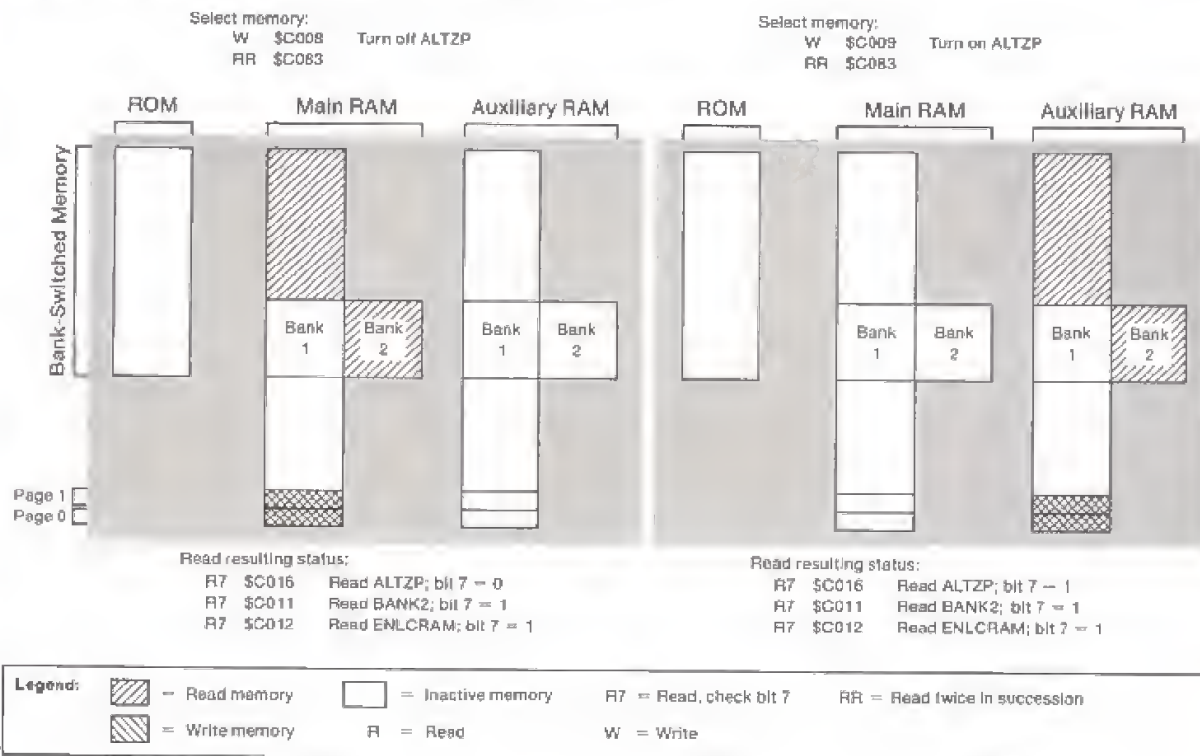


Figure 2-10. Read and Write RAM and Use Second \$D0 Bank



**Note:** You can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well. However, you can read ROM and write RAM (Figures 2-5 and 2-6), which makes it easy to transfer firmware to bank-switched RAM if you want to use it with a program there.

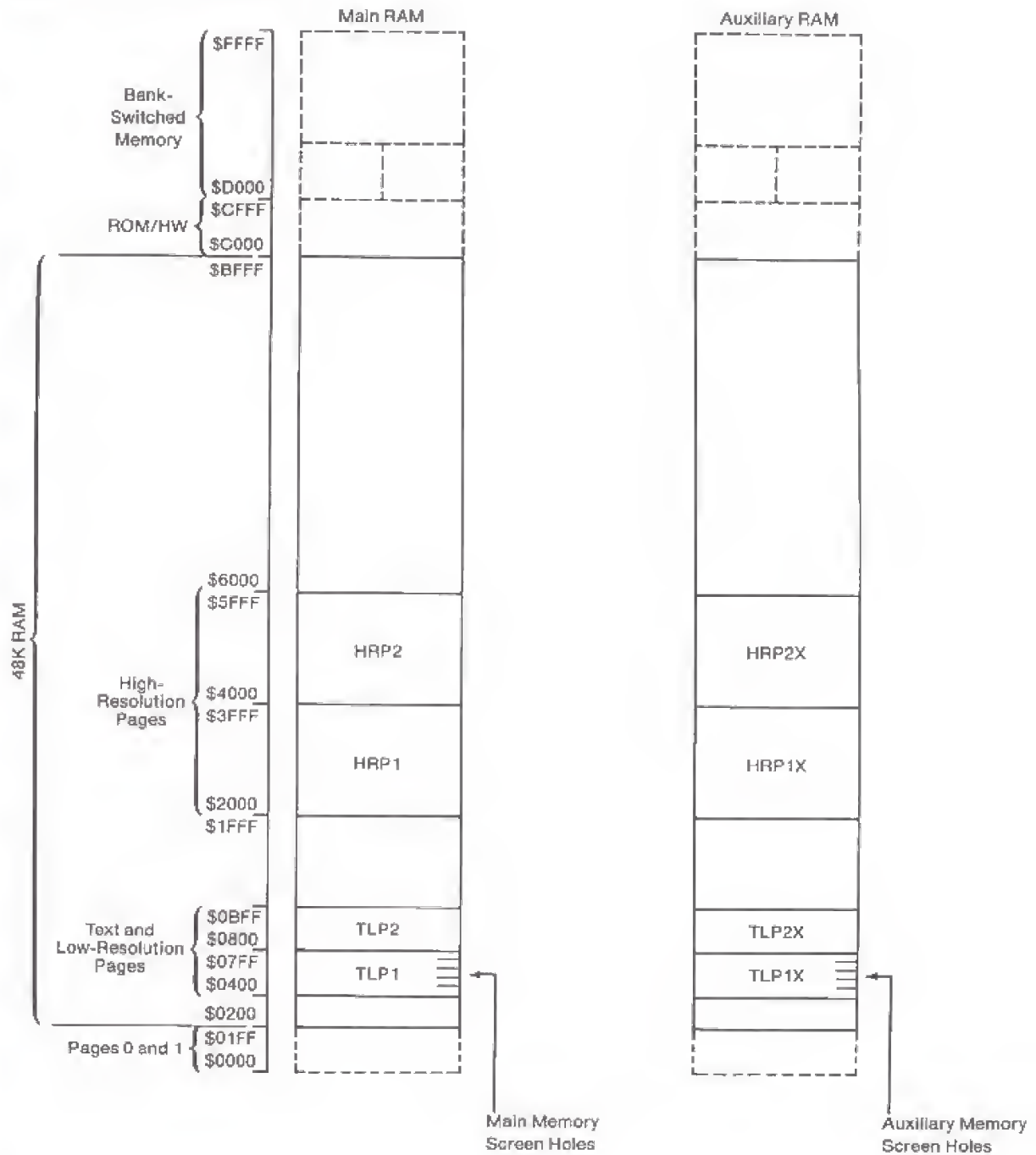
## 2.5 48K Memory

Section 2.5.1 describes the memory pages the hardware and firmware use for various purposes. Sections 2.5.2 and 2.5.4 explain how to select main or auxiliary RAM for read/write and video storage, respectively. Section 2.5.3 tells you how to use firmware routines to transfer data or program control between main and auxiliary RAM.

The 48K memory space (actually, 47-1/2K) extends from location \$200 to location \$BFFF (Figure 2-11) in both main and auxiliary RAM. The amount of storage available in this address space depends on what language or operating system you are using, and what video display needs your program has.



Figure 2-11. 48K Memory Map



The term **system** refers to the computer and its firmware plus whatever operating system you are using.

A **buffer** is any storage area set aside for one program or device to put information into and another to take information out of at a different time or rate.

Refer to Appendix D and to the appropriate programmer and reference manuals for operating system use of page 3.

**Global storage** refers to an area reserved for information that programs use in common. **Vectors**—the addresses of special routines—are examples of this kind of information. Section 2.6 discusses the global storage and vectors found on page \$03.

See Chapter 5.

See section 3.4.5.

---

### 2.5.1 Page Allocations

Most of the Apple IIc's 48K RAM is available for storing your programs and data. However, a few RAM pages are reserved for the use of the Monitor firmware, the Applesoft BASIC interpreter, and whatever video display you may select.

**Note:** The system does not prevent your using these pages, but if you do use them, you must be careful not to disturb the system data they contain.

---

#### Page \$02 (The Input Buffer)

The GETLN input routine (section 3.2.3) uses page 2 as its keyboard-input buffer. The size of this buffer (256 bytes) sets the maximum size of input strings read by Applesoft or the Monitor. If you know that you won't be typing any long input strings (more than, say, 30 characters), you can store temporary data at the upper end of page 2.

---

#### Page \$03 (Global Storage and Vectors)

The Monitor and operating systems use parts of page 3 for global storage and vectors. Table 2-7 shows the part of page 3 the built-in firmware uses.

---

#### Pages \$04 Through \$07 (Text and Low-Resolution Page 1)

The most often used display buffer is the Text and Low-Resolution Graphics Page 1 (TLP1 in Figure 2-11), which occupies main memory pages \$04 through \$07. It is not usable for program and data storage if you are using Monitor routines or Applesoft, or with almost any other program that uses text or low resolution display.

Text and Low-Resolution Page 1X (TLP1X) is an identical display page occupying auxiliary memory pages \$04 through \$07. This pair of Text and Low-Resolution graphics pages are used together to produce 80-column text display.

There are 128 locations in pages \$04-\$07 (64 in main RAM, 64 in auxiliary RAM) that are not displayed on the screen. These locations are called **screen holes**.



---

**Warning**

*The screen holes are reserved for use by the built-in firmware.*

---

---

***Pages \$08 Through \$0B (Text and Low-Resolution Page 2)***

The second Text and Low-Resolution Graphics display buffer, TLP2, occupies main memory pages \$08 through \$0B. Most programs do not use Page 2 for displays, but TLP2 is there for display use if required.

Text and Low-Resolution Page 2X (TLP2X) is an identical display buffer occupying pages \$08 through \$0B in auxiliary memory.

**Note:** Apple IIc firmware does not provide a way to use the second pair of Text and Low-Resolution graphics pages for 80-column text display.

---

***Page \$08 (Communication Port Buffers)***

Serial port 2: see Chapter 8.

Serial port 2 uses the first half of auxiliary memory page \$08 (addresses \$0800 through \$087F) as a keyboard input buffer, and the second half of the page (addresses \$0880 through \$08FF) as a serial input buffer. These buffers increase the data transfer rates possible with the serial communication port. Appendix E explains how to use these features.

If your program does not use this page for buffers, it can use it as part of TPLX.

---

***Pages \$20 Through \$3F (High-Resolution Page 1)***

The primary high-resolution-graphics display buffer, called High-Resolution Page 1 (HRP1), occupies the 32 memory pages from \$20 through \$3F (locations \$2000 through \$3FFF). If your program doesn't use high-resolution graphics, this area is usable for programs or data.

High-Resolution Page 1X (HRP1X) is an identical display page occupying auxiliary memory pages \$20 through \$3F.

See Chapter 5.

The Apple IIc can display double-high-resolution graphics by interleaving HRP1 and HRP1X.

For more information about the display buffers, see Chapter 5.

---

### ***Pages \$40 Through \$5F (High-Resolution Page 2)***

High-Resolution-Graphics Page 2 occupies main memory pages \$40 through \$5F (locations \$4000 through \$5FFF). Most programs use this area for program or data storage. However, it is also available as a second high-resolution page.

High-Resolution-Graphics Page 2X (HRP2X) occupies auxiliary memory pages \$40 through \$5F.

**Note:** Apple IIc firmware provides high-resolution graphics routines for HRP1 and HRP2 only. Refer to the *Applesoft BASIC Programmer's Reference Manual*.

---

### ***2.5.2 Using 48K Memory Switches***

Two switches select main or auxiliary RAM in the 48K memory space (Table 2-2): RAMRD determines which to use for reading, and RAMWRT determines which to use for writing. When these switches are on, they select auxiliary memory. When they are off, they select main memory.



For details, refer to section 2.5.4.

---

#### **Warning**

*This discussion assumes that the 80STORE switch, used to control display memory, is off.*

---

Each switch has three locations assigned to it: one to turn it on, one to turn it off, and a third to read its state. Because the memory locations for turning the switches on and off are shared with keyboard reading functions, you must write to these addresses to use them for memory switching.

For each switch, you can read bit 7 at its third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

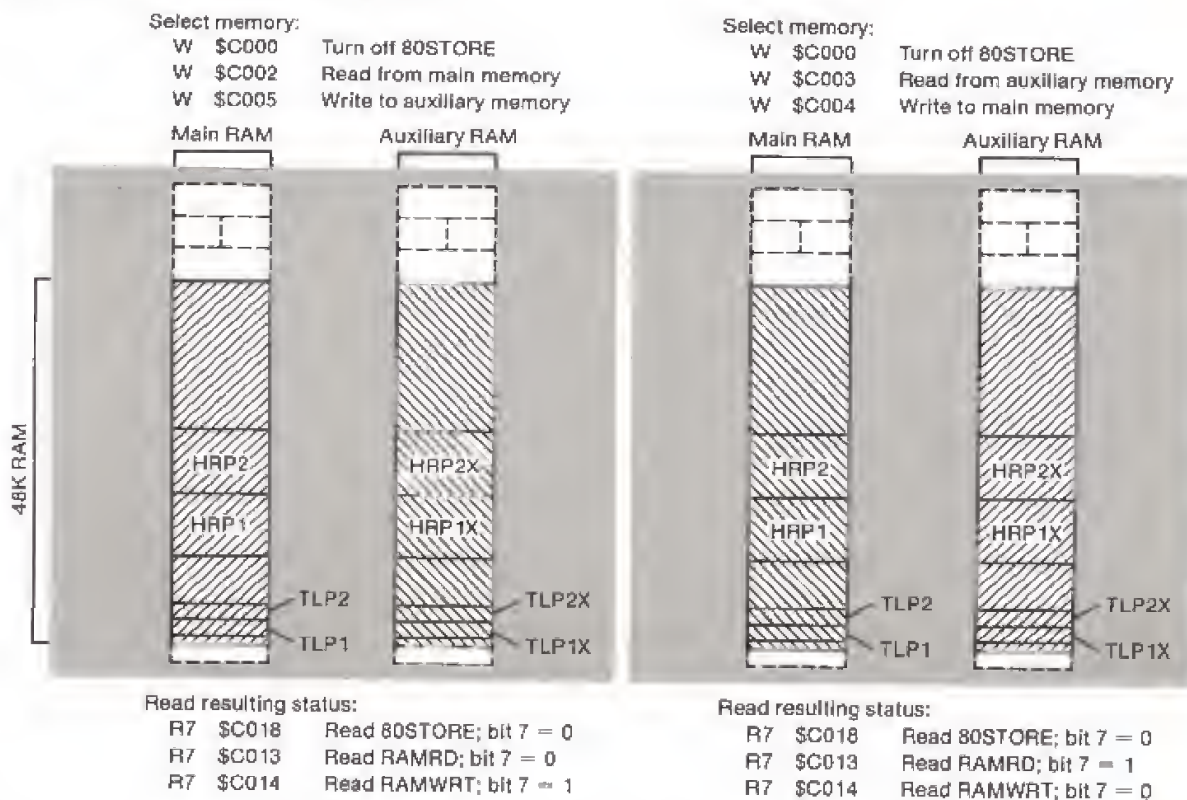
Figures 2-12 and 2-13 illustrate how the switches work.



**Table 2-2. 48K Memory Switches.** *Note: 80STORE must be off to switch all memory in this range, including display memory (Table 2-6).*

Action	Hex	Name	Function
W	\$C002	RAMRD	Off: read main 48K RAM.
W	\$C003	RAMRD	On: read auxiliary 48K RAM.
R7	\$C013	RDRAMRD	Read whether main (0) or auxiliary (1)
W	\$C004	RAMWRT	Off: write to main 48K RAM.
W	\$C005	RAMWRT	On: write to auxiliary 48K RAM.
R7	\$C014	RDRAMWRT	Read whether main (0) or auxiliary (1)

**Figure 2-12. 48K RAM Selection: Split Pairs**



**Legend:**

= Read memory

= Inactive memory

R7 = Read, check bit 7

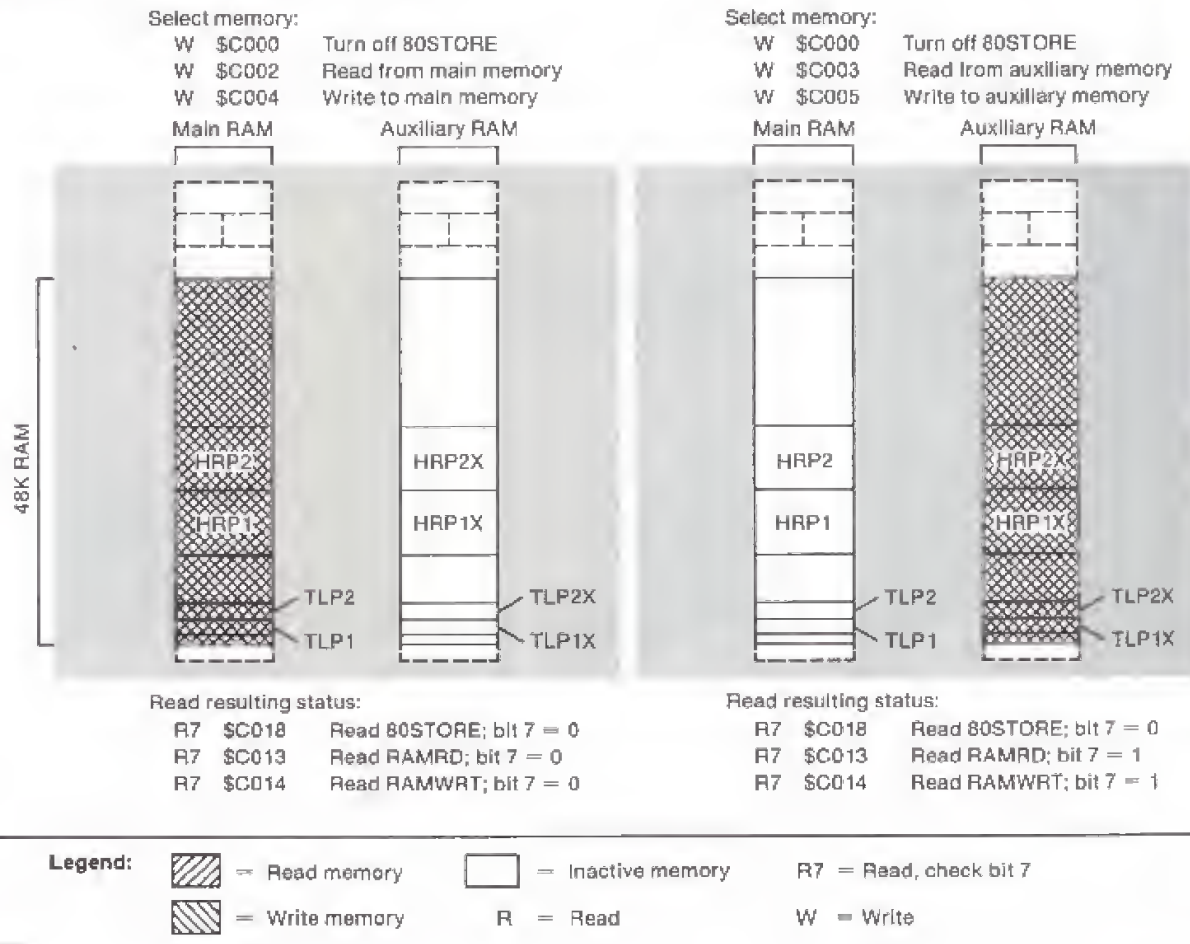
= Write memory

R = Read

W = Write



Figure 2-13. 48K RAM Selection: One Side Only



---

### 2.5.3 Transfers Between Main and Auxiliary Memory

If you want to write assembly-language programs that use auxiliary memory but you don't want to manage the auxiliary memory yourself, you can use the built-in 48K RAM transfer routines. These routines make it possible to move between main and auxiliary memory without having to manipulate the soft switches described in Section 2.5.2.

**Note:** The routines described below make it easier to use auxiliary memory, but they do not protect you from errors. You still have to plan your use of auxiliary memory to avoid catastrophic effects on your program.

*Table 2-3. 48K RAM Transfer Routines*

Action	Hex	Name	Function
JSR	\$C311	MOVEAUX	Moves data blocks between main and auxiliary 48K memory.
JMP	\$C314	XFER	Transfers program control between main and auxiliary 48K memory.

---

#### **Transferring Data**

In your assembly-language programs, you can use the built-in routine named MOVEAUX to copy blocks of data from main memory to auxiliary memory or from auxiliary memory to main memory. Before calling this routine, you must put the data addresses into byte pairs in page zero and set or clear the carry bit to select the direction of the move.

---

#### **Warning**

*Don't try to use MOVEAUX to copy data in bank-switched memory (page zero, page one or pages \$D0 through \$FF). MOVEAUX uses page zero all during the copy.*

---

The pairs of bytes you use for passing addresses to this routine are called A1, A2, and A4, and they are used for parameter passing by several of the Apple IIc's built-in routines. The addresses of these byte pairs are shown in Table 2-4.

**Table 2-4. Parameters for MOVEAUX Routine**

Name	Location	Parameter Passed
Carry		1 = Move from main to auxiliary memory 0 = Move from auxiliary to main memory
A1L	\$3C	Source starting address, low-order byte
A1H	\$3D	Source starting address, high-order byte
A2L	\$3E	Source ending address, low-order byte
A2H	\$3F	Source ending address, high-order byte
A4L	\$42	Destination starting address, low-order byte
A4H	\$43	Destination starting address, high-order byte
	X,Y,A	These registers are preserved

Put the addresses of the first and last bytes of the block of memory you want to copy into A1 and A2. Put the starting address of the block of memory you want to copy the data to into A4.

The MOVEAUX routine uses the carry bit to select the direction to copy the data. To copy data from main memory to auxiliary memory, set the carry bit (SEC instruction); to copy data from auxiliary memory to main memory, clear the carry bit (CLC instruction).

When you make the subroutine call to MOVEAUX, the subroutine copies the block of data as specified by the A register and the carry bit. When it is finished, the accumulator and the X and Y registers are just as they were when you called it.

---

### ***Transferring Control***

You can use the built-in routine named XFER to transfer control to and from program segments in auxiliary memory. You must set up three parameters before using XFER: the address of the routine you are transferring to, the direction of the transfer, and which page zero and stack you want to use.

**Table 2-5. Parameters for XFER Routine**

Name or Location	Parameter Passed
Carry	1 = Transfer from main to auxiliary memory 0 = Transfer from auxiliary to main memory
Overflow	1 = Use page zero and stack in auxiliary memory 0 = Use page zero and stack in main memory
\$3ED	Program starting address, low-order byte
\$3EE	Program starting address, high-order byte
X,Y,A	These registers are preserved.

Put the transfer address into the two bytes at locations \$3ED and \$3EE, with the low-order byte first, as usual. The direction of the transfer is controlled by the carry bit: set the carry bit to transfer to a program in auxiliary memory; clear the carry bit to transfer to a program in main memory.

Use the overflow bit to select which page zero and stack you want to use: clear the overflow bit to use the main memory; set the overflow bit (cause an overflow condition) to use the auxiliary memory.

After you have set up the parameters, pass control to the XFER routine by a jump instruction, rather than a subroutine call.



---

**Warning**

*It is your responsibility as the programmer to save the current stack pointer before using XFER and to restore it after regaining control. Failure to do so will cause program errors.*

---

Refer to Appendix E for instructions on how to do this.

---

### **2.5.4 Using Display Memory Switches**

Section 2.5.2 discusses how to select main or auxiliary RAM for the 48K memory space. However, under many circumstances your program may want to control reading and writing to display pages separately. The switches discussed in this section override the effects of RAMRD and RAMWRT for display pages only.

One of the switches, 80STORE, shares its on and off addresses with a keyboard reading function. As a result, your program must write to these locations to turn the switch on and off.

Three switches are involved in the display page selection process. Each of them has three locations assigned to it: one to turn it on, one to turn it off, and a third to read its state (Table 2-6).

For each switch, you can read bit 7 at its third location to check whether the switch is on or off. If the switch is on, bit 7 is 1; if the switch is off, bit 7 is 0.

Here is how these switches work for reading and writing.

- If HIRES is off, then PAGE2 switches between Text and Low-Resolution Graphics (TLP) pages only. If HIRES is on, then PAGE2 switches between TLP pages and High-Resolution Graphics (HRP) pages.
- If 80STORE is off, RAMRD and RAMWRT (Table 2-2) determine whether main or auxiliary RAM locations are used. PAGE2 selects pages for display (Chapter 5), but not for reading and writing.
- If 80STORE is on, it overrides RAMRD and RAMWRT with respect to the display pages selected by HIRES and PAGE2 (Figures 2-15 and 2-16).



See Chapter 5 for a discussion of text and high-resolution page and the high-resolution pages.

**Table 2-6. Display Memory Switches**

Action	Hex	Name	Function
W	\$C000	80STORE	Off: RAMRD and RAMWRT determine RAM locations.
W	\$C001	80STORE	On: PAGE2 switches between TLP1 and TLP1X, and (if HIRES on) between HRP1 and HRP1X.
R7	\$C018	RD80STORE	Read whether 80STORE on (1) or off (0)
R	\$C054	PAGE2	Off: select TLP1 and HRP1.
R	\$C055	PAGE2	On: if 80STORE off, switch to TLP2, and (if HIRES on) to HRP2. If 80STORE on, switch to TLP1X, and (if HIRES on) to HRP1X.
R7	\$C01C	RDPAGE2	Read whether PAGE2 on (1) or off (0)
R	\$C056	HIRES	Off: display text and low-resolution page.
R	\$C057	HIRES	On: display high-resolution pages; make PAGE2 switch between high-resolution pages.
R7	\$C01D	RDHIRES	Read whether HIRES on (1) or off (0)

Figure 2-14. PAGE2 Selections With 80STORE On and HIRES Off

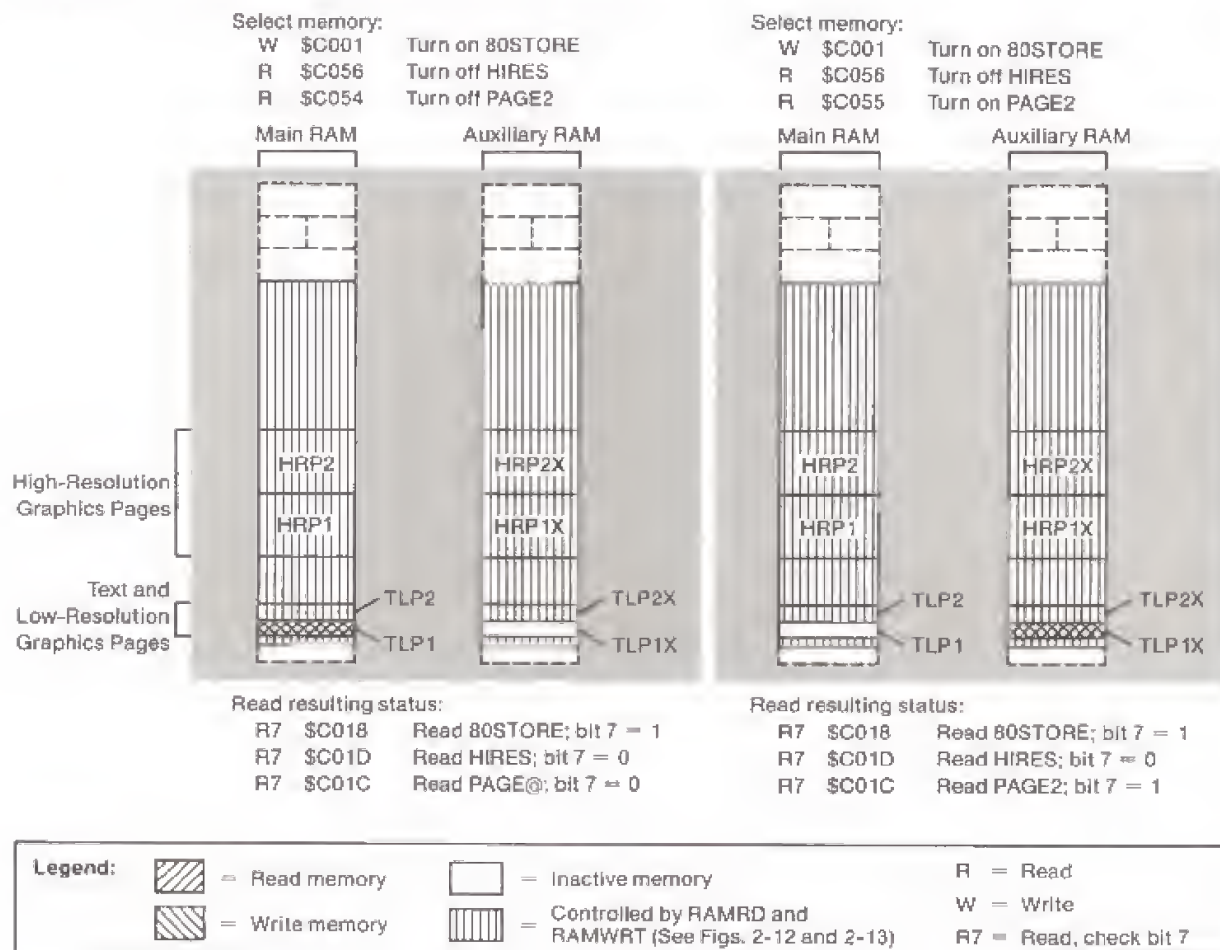
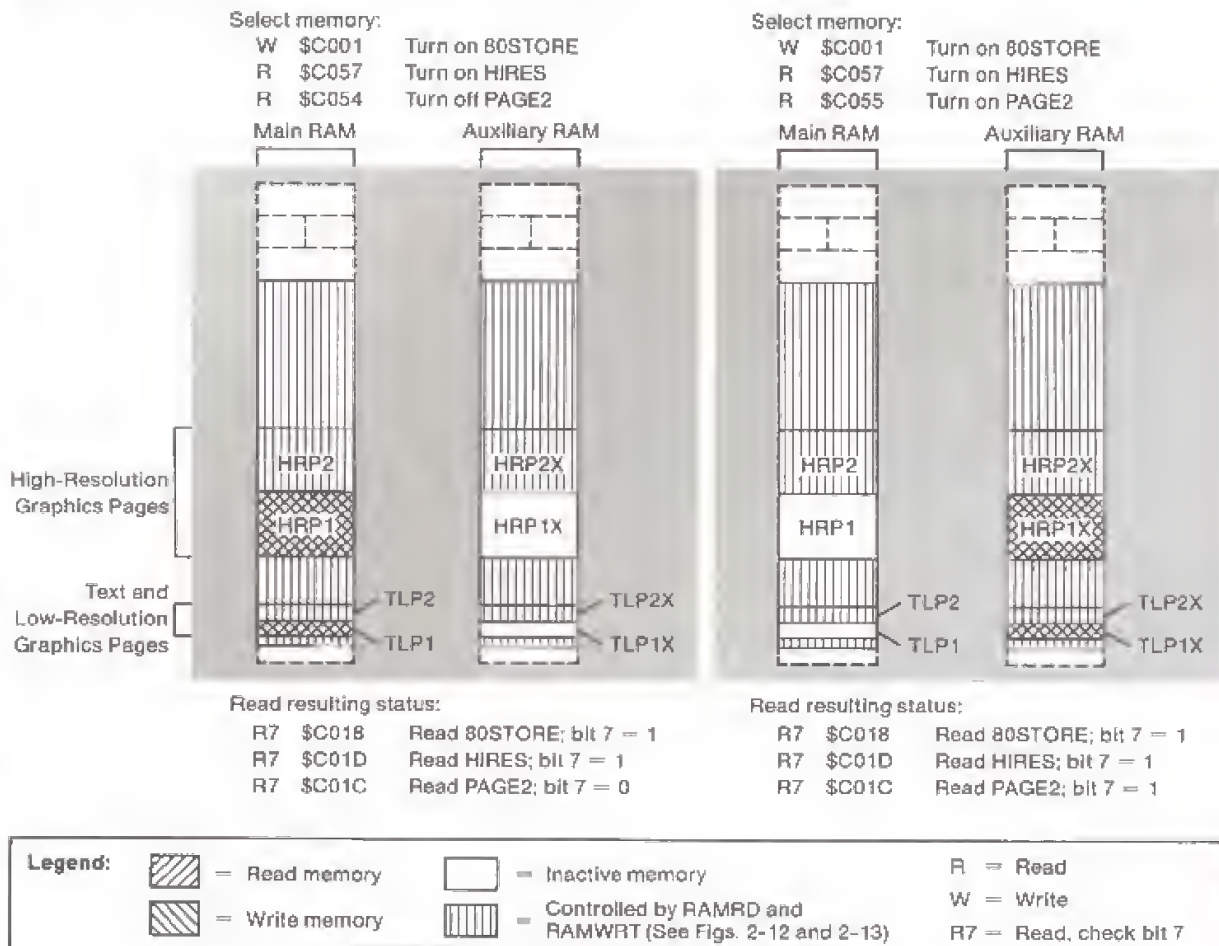


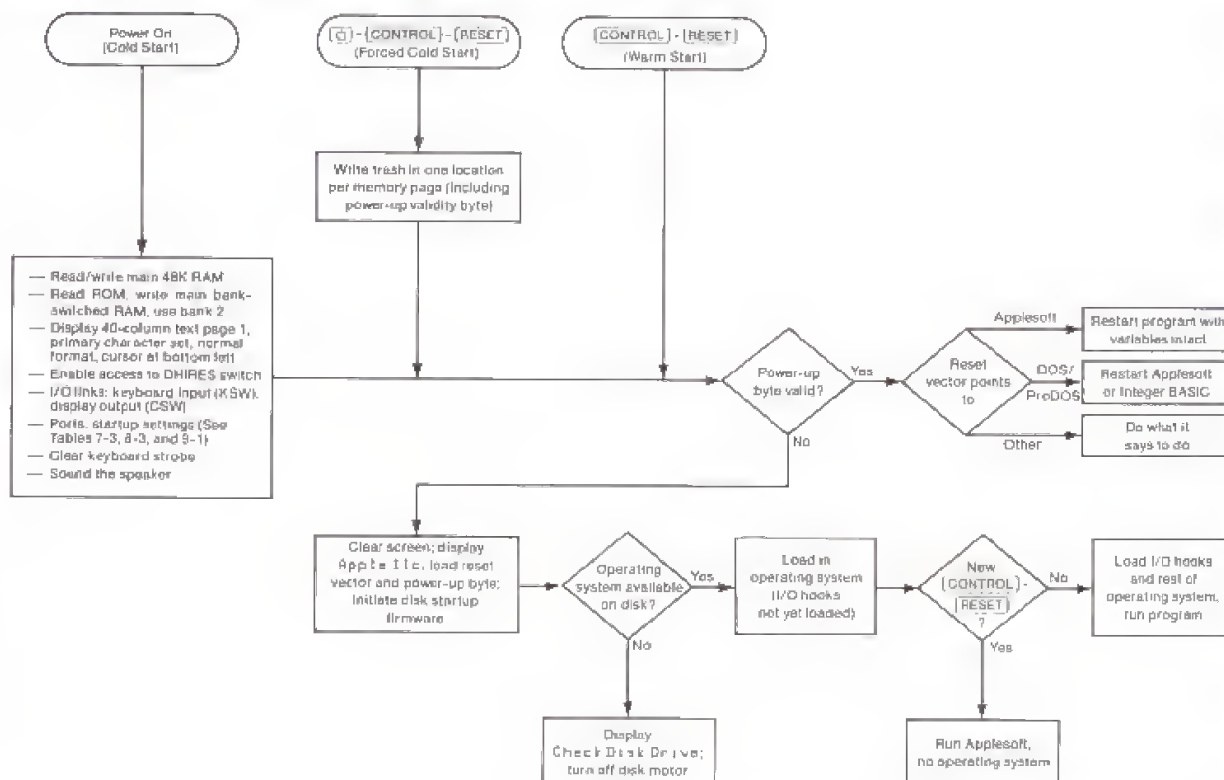
Figure 2-15. PAGE2 Selections With 80STORE On and HRES On



## 2.6 The Reset Routine

A procedure called the **reset routine** (Figure 2-16) puts the Apple IIc into a known state when it has just been turned on or you hold down **CONTROL** while pressing **RESET**. The reset routine puts the Apple IIc into its normal operating mode and restarts the program indicated at locations \$3F2 and \$3F3 (Table 2-7).

Figure 2-16. Reset Routine Flowchart



When you initiate a reset, hardware in the Apple IIc sets the memory-controlling soft switches to normal: main ROM and RAM are enabled, auxiliary RAM is disabled and the bank-switched memory space is set up to read from ROM and write to RAM, using the second bank at \$D000.

The reset vector validity check is described in section 2.6.4.

The reset routine sets the display-controlling soft switches to display 40-column text Page 1 using the primary character set, then sets the display window equal to the full 40-column display, puts the cursor at the bottom of the screen, and sets the text display format to normal.

The reset routine also sets the keyboard and display as the standard input and output devices (Chapter 3). It masks mouse interrupts and sets mouse defaults (Table 9-1). Finally, it enables DHIRES switch access (by turning on IOUDIS), clears the keyboard strobe, and sounds the speaker.

The Apple IIc has three types of reset: power-on reset, also called cold-start reset; warm-start reset; and forced cold-start reset. The procedure described above is the same for any type of reset. What happens next depends on the reset vector. The reset routine checks the reset vector to determine whether it is valid or not. If the reset was caused by turning the power on, the vector will not be valid, and the reset routine will perform the cold-start procedure. If the vector is valid, the routine will perform the warm-start procedure.

---

### ***2.6.1 The Cold-Start Procedure (Power On)***

If the reset vector is not valid, either the Apple IIc has just been turned on or something has caused memory contents to be changed. The reset routine clears the display and puts the string `Apple IIc` at the top of the display. It loads the reset vector and the validity-check byte as described in section 2.6.1, then initiates the startup routine that resides in the disk controller firmware. The bootstrap routine then loads whatever operating system resides on the disk in the built-in drive. When the operating system has been loaded, it displays other messages on the screen. If there is no disk in the disk drive, the drive motor keeps spinning for a brief time. Then the firmware shuts it off and displays the message `Check Disk Drive` at the bottom of the screen.

If you press `CONTROL-RESET` again before the startup procedure is completed, the reset routine continues without using the disk, and passes control to the Applesoft BASIC interpreter.



---

### 2.6.2 The Warm-Start Procedure (CONTROL-RESET)

Whenever you press **CONTROL-RESET** when the Apple IIc has already completed a cold-start reset, the reset vector is still valid and it is not necessary to reinitialize the entire system. The reset routine simply uses the vector to transfer control to the program it points to, which at power-up is the Applesoft interpreter.

If the vector does point to the Applesoft interpreter, your Applesoft program and variables are still intact. If you are using DOS or ProDOS™, that operating system is the resident program and it restarts the BASIC interpreter you were using when you pressed **CONTROL-RESET**.

**Note:** A program residing only in bank-switched RAM cannot use the reset vector to regain control after a reset, because upon reset the hardware selects the ROM for reading in the bank-switched memory space.





---

### 2.6.3 Forced Cold Start (⌘-CONTROL-RESET)

If a program has set the reset vector to point to its own warm-start address, as described below, pressing **CONTROL-RESET** causes transfer of control to that program. If you want to stop such a program without turning the power off and on, you can force a cold-start reset by holding down **CONTROL** and **⌘**, then pressing and releasing **RESET**.

**Note:** When you want to stop a program unconditionally—for example, to start up the Apple IIc with some other program—you should use the forced cold-start reset, **⌘-CONTROL-RESET**, instead of turning the power off and on.

The forced cold-start reset works as follows. First, it destroys the program or data in memory by writing two bytes of arbitrary data into each page of main RAM. The two bytes that get written over in page 3 are the ones that contain the reset vector. The warm-start reset routine finds the error, and so performs a normal cold-start reset.

**Note:** If you press both  and  during power-up or -, built-in exercise code is executed. This code is for production and has no end-user value.

### ***2.6.4 The Reset Vector***

The cold-start reset routine stores the starting address of the built-in Applesoft interpreter, low-order byte first, in the reset vector address at locations \$3F2 and \$3F3. It then stores a validity-check byte, also called the **power-up byte**, at location \$3F4. The validity-check byte is computed by performing an exclusive-OR of the second byte of the vector with the constant 165 (hexadecimal \$A5). Each time you reset the Apple IIc, the reset routine uses this byte to determine whether the reset vector is still valid.

You can change the reset vector so that the reset routine will transfer control to your program instead of to the Applesoft interpreter. For this to work, you must also change the validity-check byte to the exclusive-OR of the high-order byte of your new reset vector with the constant 165 (\$A5). If you fail to do this, then the next time you reset the Apple IIc, the reset routine will determine that the reset vector is invalid and perform a cold-start reset, eventually transferring control to the disk bootstrap routine or to Applesoft.

There is a subroutine that generates the validity-check byte for the current reset vector. This subroutine, called SETPWRC, is at location \$FB6F. When your program finishes, it can return the Apple IIc to normal operation by restoring the original reset vector and again calling the subroutine to fix up the validity-check byte.

*Table 2-7. Page 3 Vectors*

Vector Address	Vector Function
\$3F0 \$3F1	Address of the subroutine that handles BRK requests (normally \$59, \$FA).
\$3F2 \$3F3	Reset vector (see text).
\$3F4	Power-up byte (see text).
\$3F5 \$3F6 \$3F7	Jump instruction to the subroutine that handles Applesoft & commands (normally \$4C, \$58, \$FF).
\$3F8 \$3F9 \$3FA	Jump instruction to the subroutine that handles user <span style="border: 1px solid black; padding: 0 2px;">CONTROL</span> -Y commands.
\$3FB \$3FC \$3FD	Jump instruction to the subroutine that handles non-maskable interrupts (not used on Apple IIc).
\$3FE \$3FF	Interrupt vector (address of the subroutine that handles interrupt requests (Appendix E).



# *Introduction to Apple IIc I/O*



This chapter is an introduction to the input/output capabilities of the Apple IIc; the next six chapters discuss these capabilities in detail. The remainder of this chapter outlines the common elements of I/O processing—standard I/O links and features, standard port entry points, protocols and storage locations, and direct I/O.

### 3.1 The Standard I/O Links

When you call one of the character I/O subroutines (COUT and RDKEY), the first thing that happens is an indirect jump to an address stored in programmable memory. Memory locations used for transferring control to other subroutines are sometimes called **vectors**. In this manual, the locations used for transferring control to the I/O subroutines are called the **I/O links**.

In an Apple IIc running without an operating system, each I/O link is normally the address of the standard input or output subroutine. An operating system will typically place addresses of its own I/O routines in these link locations instead.

By calling the I/O subroutines that jump to the link addresses instead of calling the standard subroutines directly, you ensure that your program will work properly in conjunction with other software, such as the operating system or a device driver. The I/O links contain the addresses of KEYIN and COUT1 if the enhanced video firmware is off (flashing checkerboard cursor), and of C3KEYIN and C3COUT1 if that firmware is on (inverse solid cursor).

---

### 3.1.1 Changing the Standard I/O Links

The standard I/O links are two pairs of locations in the Apple IIc that are used for controlling character input and output.

**Note:** Not all operating systems use the standard I/O links. For example, Apple Pascal does not use them.

The link at locations \$36 and \$37 is called CSW, for character output switch. Individually, location \$36 is called CSWL (CSW Low) and location \$37 is called CSWH (CSW High). This link holds the starting address of the subroutine the Apple IIc is currently using for single-character output. This address is normally \$FDF0, the address of routine COUT1.

When you issue a PR#n from BASIC or an n CONTROL-P from the Monitor, the Apple IIc changes this link address to the first address in the ROM memory space allocated to port n. That address has the form \$Cn00. Subsequent calls for character output are thus transferred to the firmware starting at that address. When it has finished, the firmware executes an RTS (return from subroutine) instruction to return control to the calling program. Sometimes a PR#n will cause both input and output switches to be changed (as in the 80-column firmware).

A similar link at locations \$38 and \$39 is called KSW, for keyboard input switch. Individually, location \$38 is called KSWL (for KSW low) and location \$39 is called KSWH (KSW high). This link holds the starting address of the routine currently being used for single-character input. This address is normally \$FD1B, the starting address of the standard input routine KEYIN.

When you issue an IN#n command from BASIC or an n CONTROL-K from the Monitor, the Apple IIc changes this link address to \$Cn00, the beginning of an I/O firmware subroutine. Subsequent calls for character input are thus transferred to that firmware. The firmware puts the input character, with its high bit set, into the accumulator and executes an RTS (return from subroutine) instruction to return control to the program that requested input.

When a disk operating system (DOS or ProDOS) is running, one or both of the standard I/O links hold addresses of the disk operating system's input and output routines. The operating system has internal locations that hold the addresses of the character input and output routines that are currently active.

The Monitor is discussed in Chapter 10.



### Warning

*If a program that is running with DOS or ProDOS changes the standard link addresses, either directly or via IN# and PR# commands, the operating system may be disconnected from the system. To avoid this, BASIC programs should issue an empty PRINT statement with a carriage return (to be sure that what follows begins a new line), then another PRINT statement containing CONTROL-D and the IN# or PR# command.*

Refer to the section on input and output link addresses in the operating system manuals for further details.

After changing CSW or KSW, assembly-language programs running under DOS should call the subroutine at location \$3EA. This subroutine transfers the link address to a location inside the operating system and then restores the operating system link address in the standard link location.

## 3.2 Standard Input Features

GETLN also provides on-screen editing features: see section 3.2.5.

The Apple IIc's firmware includes two different subroutines for reading from the keyboard. One subroutine is named RDKEY (*read key*). It calls the current character input routine (that is, the one whose address is stored at KSW). This is normally KEYIN or C3KEYIN, which accepts one character from the keyboard. The other subroutine is named GETLN (*get line*). GETLN accepts a sequence of characters terminated with a carriage return. Thus GETLN allows line-oriented input using the current input routine.

### 3.2.1 RDKEY Input Subroutine

A program gets a character from the keyboard by making a subroutine call to RDKEY at memory location \$FD0C. RDKEY passes control via the input link KSW to the current input subroutine, which is normally KEYIN.

RDKEY displays a cursor at the current cursor position, which is immediately to the right of whatever character you last sent to the display (normally by using the COUT routine, described below).

---

### 3.2.2 KEYIN Input Subroutine

KEYIN is the standard input subroutine. When called, it displays a cursor, waits until the user presses a key, then returns to the calling program with the ASCII code of the key pressed in the accumulator.

If the enhanced video firmware is inactive, KEYIN displays a cursor by alternately storing a checkerboard block in the cursor location, then storing the original character, then the checkerboard again. If the firmware is active, C3KEYIN inverts the character at the cursor position.

KEYIN also generates a random number. While it is waiting for the user to press a key, KEYIN repeatedly increments the 16-bit number in memory locations \$4E and \$4F. This number keeps increasing from 0 to \$FFFF (65535), then starts over again at 0. The value of this number changes so rapidly that there is no way to predict what it will be after a key is pressed. A program that reads from the keyboard can use this value as a random number or as a seed for a pseudo-random number routine.

When the user presses a key, KEYIN accepts the character, stops displaying the cursor, and returns to the calling program with the character in the accumulator.

---

### 3.2.3 GETLN Input Subroutine

Programs often need strings of characters as input. While it is possible to call RDKEY repeatedly to get several characters from the keyboard, there is a more powerful subroutine you can use. This routine is named GETLN, which stands for *get line*, and it starts at location \$FD6A. Using repeated calls to RDKEY, GETLN accepts characters from the standard input subroutine—usually KEYIN—and puts them into the input buffer located in the memory page from \$200 to \$2FF. GETLN also provides the user with on-screen editing and control features.

The first thing GETLN does when you call it is to display a prompting character, called simply a **prompt**. The prompt indicates to the user that the program is waiting for input. Different programs use different prompt characters, helping to remind the user which program is requesting the input. For example, an INPUT statement in a BASIC program displays a question mark ( ? ) as a prompt. The prompt characters used by the different programs on the Apple IIc are shown in Table 3-1.

Section 3.2.5 describes other features of GETLN.



GETLN uses the character stored at memory location \$33 as the prompt character. In an assembly-language program, you can change the prompt to any character you wish. In BASIC, changing the prompt character has no effect, because both BASIC interpreters and the Monitor restore it each time they request input from the user.

**Note:** Applesoft uses GETLN1 (\$FD6F) when a program is executing. GETLN1 does not print a prompt.

*Table 3-1. Prompt Characters*

Prompt Character	Program Requesting Input
?	User's BASIC program (INPUT statement)
]	Applesoft BASIC (Appendix D)
>	Integer BASIC (Appendix D)
*	Firmware Monitor (Chapter 10)

Control characters echoed by GETLN are not executed.

As the user types each character, GETLN sends the character to the standard output routine—normally COUT1—which displays it at the current cursor position and then advances the cursor to indicate the next character position.

GETLN stores the characters in its buffer, starting at memory location \$200 and using the X register to index the buffer. GETLN continues to accept and display characters until the user presses **RETURN** (or **CONTROL-X** to cancel the line). Then it clears the remainder of the line the cursor is on, stores the carriage-return code to mark the end of the buffer, places the cursor at the beginning of the next line, and returns.

The maximum line-length that GETLN can handle is 255 characters. If the user types more than this, GETLN sends a backslash (\) and a carriage return to the display, cancels the line it has accepted so far, and starts over. To warn the user that the line is getting full, GETLN sounds a bell (tone) at every keypress after the 248th.

**Note:** The Applesoft interpreter accepts only 239 characters.



### 3.2.4 Escape Codes With GETLN

GETLN has many special functions that you invoke by typing escape codes on the keyboard. An escape code is obtained by pressing (ESC), releasing it, and then pressing some other key, as shown in Table 3-2.

**Note:** Be sure to release (ESC) right away. If you hold it too long, the auto-repeat mechanism will begin, which may cancel the ESC.

In escape mode, you can keep using the arrow keys and the cursor-motion keys (I), (J), (K), and (M) without pressing (ESC) again. This enables you to perform repeated cursor moves by holding down the appropriate key.

When GETLN is in escape mode, it displays an inverse plus sign as the cursor. You leave escape mode by typing any key other than a cursor-motion key.

**Note:** The escape codes with the arrow keys are the standard cursor-motion keys on the Apple IIc. The escape codes with (I), (J), (K), and (M) are the standard cursor-motion keys on the Apple II and II Plus, and are present on the Apple IIc for compatibility.

**Table 3-2.** Escape Codes With GETLN. (1) Cursor-control key; see text. (2) This code functions only when the enhanced video firmware is active.

Escape Code	Function	Notes
(ESC) (H)	Clears the window and homes the cursor (places it in the upper-left corner of the screen), then exits from escape mode.	
(ESC) (A) or (a)	Moves the cursor right one line and exits from escape mode.	
(ESC) (B) or (b)	Moves the cursor left one line and exits from escape mode.	
(ESC) (C) or (c)	Moves the cursor down one line and exits from escape mode.	
(ESC) (D) or (d)	Moves the cursor up one line; exits from escape mode.	
(ESC) (E) or (e)	Clears to the end of the line; exits from escape mode.	
(ESC) (F) or (f)	Clears to the bottom of the window; exits from escape mode.	

**Table 3-2—Continued. Escape Codes With GETLN**

Escape Code	Function	Notes
<code>(ESC) (I) or (i)</code> or <code>(ESC) (7)</code>	Moves the cursor up one line; remains in escape mode.	1
<code>(ESC) (J) or (j)</code> or <code>(ESC) (~)</code>	Moves the cursor left one space; remains in escape mode.	1
<code>(ESC) (K) or (x)</code> or <code>(ESC) (-)</code>	Moves the cursor right one space; remains in escape mode.	1
<code>(ESC) (M) or (m)</code> or <code>(ESC) (L)</code>	Moves the cursor down one line; remains in escape mode.	1
<code>(ESC) (4)</code>	Switches to 40-column mode; activates the enhanced video firmware; sets links to C3KEYIN and C3COUT1; restores normal window size (Table 3-5); exits from escape mode.	2
<code>(ESC) (8)</code>	Switches to 80-column mode; activates the enhanced video firmware; sets links to C3KEYIN and C3COUT1; restores normal window size (Table 3-5); exits from escape mode.	2
<code>(ESC) (CONTROL) (D)</code>	Disables control characters; only carriage return, line feed, BELL, and backspace have an effect when printed.	
<code>(ESC) (CONTROL) (E)</code>	Reactivates control characters.	
<code>(ESC) (CONTROL) (Q)</code>	Deactivates the enhanced video firmware; sets links to KEYIN and COUT1; restores normal window size (Table 3-5); exits from escape mode.	2

Escape sequences can be used in the middle of an input line to change the appearance of the screen. They have no effect on the input line.

For an introduction to editing with these features, refer to the *Applesoft Tutorial*.

---

### 3.2.5 Editing With GETLN

Subroutine GETLN provides the standard on-screen editing features used by the BASIC interpreters and the Monitor. Any program that uses GETLN for reading the keyboard has these features.

---

#### Cancel Line

Any time you are typing a line, pressing **(CONTROL)-(X)** causes GETLN to cancel the line. GETLN displays a backslash ( \ ) and issues a carriage return, then displays the prompt and waits for you to type a new line. GETLN takes the same action when you type more than 255 characters, as described above.

---

#### Backspace

When you press **(←)** (or **(CONTROL)-(H)**), GETLN moves its buffer pointer back one space, effectively deleting the last character in its buffer. It also sends a backspace character to routine COUT, which moves the cursor back one space. If you type another character now, it will replace the character you backspaced over, both on the display and in the line buffer.

Each time you press **(←)**, it moves the cursor left and deletes another character, until you are back at the beginning of the line. If you then press **(←)** one more time, you have effectively canceled the line, and GETLN issues a carriage return and displays the prompt.

---

#### Retype

**(→)** (or **(CONTROL)-(U)**) has a function that is complementary to the backspace function. When you press **(→)**, GETLN picks up the character under the cursor just as if it had been typed on the keyboard. You can use this procedure to pick up characters that you just deleted by backspacing across them. You can use the backspace and retype functions with the cursor-motion functions to edit data on the display.

The cursor moves even if the deleted character is an (invisible) control character. Thus it is possible for screen alignment and buffer alignment to be different.

See section 3.2.4.

## 3.3 Standard Output Features

The standard output routine is named COUT (pronounced *C-out*) which stands for character output. COUT normally calls COUT1 or C3COUT1, which sends one character to the display, advances the cursor position, and scrolls the display when necessary. COUT1 and C3COUT1 restrict their use of the display to an active area called the **text window**, described below.

### 3.3.1 COUT Output Subroutine

Your program makes a subroutine call to COUT at memory location \$FDED with a character in the accumulator. COUT then passes control via the output link CSW to the current output subroutine, normally COUT1 or C3COUT1, which takes the character in the accumulator and writes it out. If the accumulator contains an uppercase or lowercase letter, a number, or a special character, COUT1 or C3COUT1 displays it; if the accumulator contains a control character, COUT1 or C3COUT1 either performs one of the special functions described below or ignores the character.

Each time you send a character to COUT1 or C3COUT1, it displays the character at the current cursor position, replacing whatever was there, and then advances the cursor position one space to the right. If the cursor position is already at the right-hand edge of the window, COUT1 or C3COUT1 moves it to the leftmost position on the next line down. If this would move the cursor position past the end of the last line in the window, COUT1 or C3COUT1 scrolls the display up one line and sets the cursor position at the left end of the new bottom line.

The cursor position is controlled by the values in memory locations \$24 and \$25. These locations are named CH, for cursor horizontal, and CV, for cursor vertical. COUT1 and C3COUT1 do not display a cursor, but the input routines described above do, and they use this cursor position. However, changing CV directly will not change the cursor's vertical position until the next carriage return or reaching the end of the current line causes a call to VTAB (for setting the base address within windows). If some other routine displays a cursor, it will not necessarily put it in the cursor position used by COUT1 or C3COUT1.



### Warning

*When the video firmware is set for 80-column display, the value of CH is kept at 0 and the true horizontal position is stored at \$57B. When the 80-column video firmware is active, use \$57B instead of CH.*

Escape codes are described in section 3.2.4.

### 3.3.2 Control Characters With COUT1

COUT1 does not display control characters. Instead, the control characters listed in Table 3-3 are used to initiate some action by the firmware. Other control characters are ignored. Most of the functions listed here can also be invoked from the keyboard, either by typing the control character listed or by using the appropriate escape code. The stop-list function, described separately, can only be invoked from the keyboard.

*Table 3-3. Control Characters With COUT1.*

Control Character	ASCII Name	Apple IIc Name	Action Taken by COUT1
CONTROL-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.
CONTROL-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.
CONTROL-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.
CONTROL-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.

### 3.3.3 Control Characters With C3COUT1

C3COUT1 does not display control characters. Instead, the control characters listed in the two parts of Table 3-4 are used to initiate some action by the firmware. Other control characters are ignored. Most of the functions listed here can also be invoked by using the appropriate escape code. The stop-list function, described separately, can only be invoked from the keyboard.

Escape Codes: see section 3.2.4.



**Table 3-4. Control Characters With C3COUT1.** (1) Only available when enhanced video firmware is active. (2) Only works from the keyboard. (3) Doesn't work from the keyboard.

Control Character	ASCII Name	Apple IIc Name	Action Taken by C3COUT1	Notes
CONTROL-G	BEL	bell	Produces a 1000 Hz tone for 0.1 second.	
CONTROL-H	BS	backspace	Moves cursor position one space to the left; from left edge of window, moves to right end of line above.	
CONTROL-J	LF	line feed	Moves cursor position down to next line in window; scrolls if needed.	
CONTROL-K	VT	clear EOS	Clears from cursor position to the end of the screen.	1,3
CONTROL-L	FF	home and clear	Moves cursor position to upper-left corner of window and clears window.	1,3
CONTROL-M	CR	return	Moves cursor position to left end of next line in window; scrolls if needed.	
CONTROL-N	SO	normal	Sets display format normal.	1,3
CONTROL-O	SI	inverse	Sets display format inverse.	1,3
CONTROL-Q	DC1	40-column	Sets display to 40-column.	1,3
CONTROL-R	DC2	80-column	Sets display to 80-column.	1,3
CONTROL-S	DC3	stop-list	Stops listing characters on the display until another key is pressed.	2
CONTROL-U	NAK	quit	Deactivates enhanced video firmware.	1,3
CONTROL-V	SYN	scroll	Scrolls the display down one line, leaving the cursor in the current position.	1,3
CONTROL-W	ETB	scroll-up	Scrolls the display up one line, leaving the cursor in the current position.	1,3
CONTROL-X	CAN	disable MouseText	Disable MouseText character display; use inverse uppercase.	
CONTROL-Y	EM	home	Moves cursor position to upper-left corner of window (but doesn't clear).	1,3
CONTROL-Z	SUB	clear line	Clears the line the cursor position is on.	1,3

*Table 3-4—Continued. Control Characters With C3COUT1*

Control Character	ASCII Name	Apple IIc Name	Action Taken by C3COUT1	Notes
CONTROL- <code>[</code>	ESC	enable MouseText	Map inverse uppercase characters to MouseText characters.	
CONTROL- <code>\</code>	FS	fwd. space	Moves cursor position one space to the right; from right edge of window, moves it to left end of line below.	1,3
CONTROL- <code>]</code>	GS	clear EOL	Clears from the current cursor position to the end of the line (that is, to the right edge of the window).	1,3
CONTROL- <code>_</code>	US	up	Moves cursor up a line, no scroll.	

### **3.3.4 The Stop-List Feature**

When you are using any program that displays text via COUT1 or C3COUT1, you can make it stop updating the display by pressing `(CONTROL)-(S)` (that is, by holding down `(CONTROL)` while pressing `(S)`). Whenever COUT1 or C3COUT1 gets a carriage return from the program, it checks for `(CONTROL)-(S)`. If it has been pressed, COUT1 or C3COUT1 stops and waits for another keypress. Then it continues normally. The character code of the key you pressed to resume displaying is ignored unless it is `(CONTROL)-(G)`. COUT1 or C3COUT1 passes CONTROL-C back to the program; if it is a BASIC program, this enables you to terminate the program while in stop-list mode.

### **3.3.5 The Text Window**

After you start up the computer or perform a reset, the firmware uses the entire display. However, you can restrict video activity to any rectangular portion of the display you wish. The active portion of the display is called the text window. COUT1 or C3COUT1 puts characters only into the window; when it reaches the end of the last line in the window, it scrolls only the contents of the window.

You can set the top, bottom, left side, and width of the text window by storing the appropriate values into four locations in memory. This enables your programs to control the placement of text in the display and to protect other portions of the screen from being written over by new text.

Memory location \$20 contains the number of the leftmost column in the text window. This number is normally 0, the number of the leftmost column in the display. In a 40-column display, the maximum value for this number is 39 (hexadecimal \$27); in an 80-column display, the maximum value is 79 (hexadecimal \$4F).

Memory location \$21 holds the width of the text window. For a 40-column display, this value is normally 40 (hexadecimal \$28); for an 80-column display, it is normally 80 (hexadecimal \$50).



---

**Warning**

*Be careful not to let the sum of the window width and the leftmost position in the window exceed the width of the display you are using (40- or 80-columns). If this happens, it is possible for COUT1 or C3COUT1 to put characters into memory locations outside the display page, possibly destroying programs or data.*

---

Memory location \$22 contains the number of the top line of the text window. This is normally 0, the topmost line in the display. Its maximum value is 23 (hexadecimal \$17).

Memory location \$23 contains the number of the bottom line of the screen, plus 1. It is normally 24 (hexadecimal \$18) for the bottom line of the display. Its minimum value is 1.

Table 3-5 summarizes the memory locations and the possible values for the window parameters.



---

**Warning**

*Pascal does not use this method of supporting window widths.*

---

**Table 3-5. Text Window Memory Locations**

Window Parameter	80col.		Minimum Location		Normal Values: Value		40col.		Maximum Values: 40col.		80col.	
	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
Left Edge	32	\$20	0	\$0	0	\$0	0	\$0	39	\$27	79	\$4F
Width	33	\$21	0	\$0	40	\$28	80	\$50	40	\$28	80	\$50
Top Edge	34	\$22	0	\$0	0	\$0	0	\$0	23	\$17	23	\$17
Bottom Edge	35	\$23	1	\$1	24	\$18	24	\$18	24	\$18	24	\$18

Both these display character sets are described in Chapter 5.

### **3.3.6 Normal, Inverse, and Flashing Text**

The form of a displayed character depends on two things: what value is stored in zero page location \$32 (the inverse flag), and whether the enhanced video firmware is off or on. The effects of the inverse flag are discussed in the next two subsections.

If the enhanced video firmware is off, the Apple IIc displays what is called the primary character set; if the video firmware is on, the Apple IIc displays what is called the alternate character set.

The primary character set includes normal (light on dark), inverse (dark on light), and flashing (alternating normal and inverse) characters. Lowercase inverse characters are not included in this set.

The alternate character set includes normal and inverse characters (including lowercase inverse), and a set of icons called MouseText. Flashing characters are not included in this set.

To display a character, load it in the accumulator, and then jump to the character-output subroutine COUT. For example, to display the character corresponding to \$C8

```
LDA #$C8
JSR COUT
```

#### **Primary Character Set Display**

Subroutine COUT1 (the standard output link when enhanced video firmware is off) can display text in normal, inverse, or flashing format, but cannot display inverse or flashing *lowercase* text.

For a brief explanation of logical functions, refer to Appendix H.

If the value of the character is greater than or equal to \$A0, the value is logically ANDed with the value of the inverse flag (at location \$32), then displayed.

If the inverse flag value is 255 (hexadecimal \$FF), the character is displayed in normal format; if the value is 63 (hexadecimal \$3F), the character is displayed in inverse format. If the value is 127 (hexadecimal \$7F) the character is displayed in flashing format.

**Note:** To avoid unusual character display results, use only the three values \$3F, \$7F and \$FF.

Character values from \$80 through \$9F are interpreted as control characters and are executed, if possible.

Character values from \$00 through \$7F are all display characters, not control characters.

---

#### ***Alternate Character Set Display***

Subroutine C3COUT1 (the standard output link when the enhanced video firmware is active) can display characters in normal or inverse format, and can display a set of icons called MouseText.

If the character is in the range \$00 through \$1F or \$80 through \$9F, it is interpreted as a control character and not displayed. Values \$20 through \$7F and \$A0 through \$FF are displayed.

If inverse flag (location \$32) bit 7 is 1, the character value is left alone. If inverse flag bit 7 is 0, the character value is ANDed with \$7F (turning off bit 7) to make it display as an inverse character.

If MouseText has not been turned on, then the values \$40 through \$5F are mapped to values \$00 through \$1F, so they display as the inverse uppercase set. If MouseText has been turned on, the values \$40 through \$5F are left unchanged, and they display as MouseText icons.

---

#### **Warning**

*Use only \$3F, \$7F or \$FF in location \$32. Other values will cause unpredictable results.*

---

MouseText: see Chapter 5.

See section 5.2.2.





## 3.4 Port I/O

Apple IIc is a member of the Apple II family of computers; however, unlike the Apple II, II Plus and IIe, the Apple IIc does not have peripheral connector slots. In place of these, it has ports—the equivalent of firmware interface cards installed in slots.

### 3.4.1 Standard Link Entry Points

To maintain compatibility with existing software and its protocols, each port's I/O firmware has the same standard entry points (\$Cn00) as its equivalent slot would have. Table 3-6 shows these equivalents, as well as listing the chapter where each port is described.

Section 3.1 describes under what conditions these entry addresses are placed in CSW and KSW. For example, issuing PR#n or IN#n changes the output and input links, respectively, so that subsequent output or input is handled by the firmware starting at address \$Cn00, and thus goes to or comes from the selected device.

*Table 3-6. Port Characteristics*

Port	Entry Point	Port Connector	Use	Chapter
1	\$C100	Serial port 1	Printers	7
2	\$C200	Serial port 2	Communication	8
3	\$C300	Video connectors	Enhanced video firmware	5
4	\$C400	Mouse	Mouse	9
5	\$C500	Reserved		
6	\$C600	Disk drives	Built-in and external drives	6
7	\$C700	No device	External drive startup (under ProDOS only)	6

**Note:** The addresses shown in Table 3-6 are not entry points in the sense that you can send characters to be printed by sending them to JSR \$Cn00.

---

### 3.4.2 Firmware Protocol

Besides the standard link address, there is also a standard firmware protocol that provides a table of device identification and entry points to standard and optional firmware subroutines (Table 3-7).

Each table begins with identification bytes. Then, starting with address  $\$Cn0D$ , each byte in the table represents the low-order byte of the entry-point address of a firmware routine. The high-order byte of the address is  $\$Cn$ , where  $n$  is the port number. Using these byte values, a program can construct its own jump table for subroutine calls.

On entry, all routines require that the X register contain  $\$Cn$  ( $n$  is the port number), and that the Y register contain  $\$n0$ .

On exit, all routines return an error code in the X register (0 means no error occurred; 3 means the request was invalid). The carry bit in the program status register usually contains a reply to a request code (0 means *no*; 1 means *yes*).

All of the Apple IIc ports except the disk port conform to this protocol.

**Table 3-7. Firmware Protocol Locations**

Address	Value	Description
\$Cn05	\$38	Pascal firmware card/port identifier
\$Cn07	\$18	Pascal firmware card/port identifier
\$Cn0B	\$01	Generic signature byte of a firmware card/port
\$Cn0C	\$cl	Device signature byte: <i>i</i> is an identifier (not necessarily unique).
		<i>c</i> = device class (not all used on the Apple IIc):
	\$0	reserved
	\$1	printer
	\$2	hand control or other X-Y device
	\$3	serial or parallel I/O card/port
	\$4	modem
	\$5	sound or speech device
	\$6	clock
	\$7	mass-storage device
	\$8	80-column card/port
	\$9	network or bus interface
	\$A	special purpose (none of the above)
	\$B-F	reserved
\$Cn0D	ii	\$Cnii is the initialization entry address (PINIT).
\$Cn0E	rr	\$Cnrr is the read routine entry address (PREAD). (Returns character read in A register)
\$Cn0F	ww	\$Cnww is the write routine entry address (PWRITE). (Enter with character to write in A register)
\$Cn10	ss	\$Cnss is the status routine entry address (PSTATUS). (Enter with request code in A register: 0 to ask "Are you ready to accept output?" or 1 to ask "Do you have input ready?")
\$Cn11		\$00 if additional address bytes follow; nonzero if not

For more information, refer to the hardware page memory map in Appendix B.

---

### 3.4.3 Port I/O Space

By a convention used in other Apple II series machines, each port or slot has exclusive use of sixteen memory locations of the form  $\$C080 + \#n0$ , where  $n$  is the port or slot number. These locations are set aside for data input and output. Table 3-8 lists the port I/O space used in the Apple IIc.

*Table 3-8. Port I/O Locations*

Port	Locations
1	$\$C090-\$C09F$
2	$\$C0A0-\$C0AF$
6	$\$C0E0-\$C0EF$

---

### 3.4.4 Port ROM Space

In the Apple II and IIe, one 256-byte page of memory space is allocated to each slot. This space is used for read-only memory (ROM or PROM) with driver programs that control the operation of input/output devices, as outlined in Table 3-7. On the Apple IIc, this space is dedicated to port firmware. However, I/O ROM space in the Apple IIc is used as efficiently as possible, and so there is not a strict correspondence between firmware for port  $n$  and the  $\$Cn00$  space, except as regards entry points.

---

### 3.4.5 Expansion ROM Space

The 2K-byte memory space from  $\$C800$  to  $\$CFFF$  in the Apple IIc—called expansion ROM space on Apple II, II Plus and IIe—contains the enhanced video firmware and port and memory transfer subroutines. Unlike the Apple II, II Plus, or IIe, the Apple IIc always has this space switched in.

---

### 3.4.6 Port Screen-Hole RAM Space

There are 128 bytes of memory (64 in main memory, 64 in auxiliary memory) allocated to the ports, eight bytes per port, as shown in Table 3-9. These bytes are reserved for use by the system, except as described in Chapters 4 through 9.

These addresses are unused bytes in the RAM memory reserved for text and low-resolution graphics displays, and hence they are sometimes called **screen holes**. These particular

locations are not displayed on the screen and their contents are not changed by the built-in output routines. In other words, they are used by the output routines but they are not part of the video display.



---

**Warning**

*All the screen holes in auxiliary memory, and many of them in main memory, are reserved for special use by Apple IIc firmware—for example to store initialization information. Do not use any locations marked reserved in this manual.*

---



*Table 3-9. Port Screen-Hole Memory Locations*

Base Address	Ports						
	1	2	3	4	5	6	7
\$0478	\$0479	\$047A	\$047B	\$047C	\$047D	\$047E	\$047F
\$04F8	\$04F9	\$04FA	\$04FB	\$04FC	\$04FD	\$04FE	\$04FF
\$0578	\$0579	\$057A	\$057B	\$057C	\$057D	\$057E	\$057F
\$05F8	\$05F9	\$05FA	\$05FB	\$05FC	\$05FD	\$05FE	\$05FF
\$0678	\$0679	\$067A	\$067B	\$067C	\$067D	\$067E	\$067F
\$06F8	\$06F9	\$06FA	\$06FB	\$06FC	\$06FD	\$06FE	\$06FF
\$0778	\$0779	\$077A	\$077B	\$077C	\$077D	\$077E	\$077F
\$07F8	\$07F9	\$07FA	\$07FB	\$07FC	\$07FD	\$07FE	\$07FF

Port firmware use of these RAM locations and their assigned hardware addresses appear in the six chapters that follow this one.

### 3.5 Interrupts

Appendix E gives a full description of interrupt handling on the Apple IIc.

When the IRQ line on the 65C02 microprocessor is activated, the 65C02 transfers control through the vector in locations \$FFFE-\$FFFF of ROM or whichever bank of RAM is switched in (Chapter 2). If ROM is switched in, this vector is the address of the Monitor's interrupt handler, which determines whether the request is due to an interrupt that should be handled internally. If so, the Monitor handles it and then returns control to the interrupted program.

If the interrupt is due to a BRK (\$00) instruction, control is transferred through the BRK vector (\$3F0-\$3F1). Otherwise, control is transferred through the IRQ vector (\$3FE-\$3FF).

# *Keyboard and Speaker*

This chapter describes how to use two of the Apple IIc's built-in devices: the keyboard and the speaker.

## 4.1 Keyboard Input

Table 4-1 describes the overall characteristics of the keyboard. Monitor keyboard support includes the three standard input routines described in Chapter 3.

*Table 4-1. Keyboard Input Characteristics*

<b>Port Number</b>	None
<b>Commands</b>	Keyboard is always on, in the sense that any keypress generates a KSTRB.
<b>Initial Characteristics</b>	Reset routine clears the keyboard strobe and sets the keyboard as the standard input device (that is, sets KSW to point to RDKEY).
<b>Hardware Locations</b>	<b>Description</b>
\$C000	Keyboard data and strobe
\$C010	Any-key-down flag and Clear-strobe switch
\$C060	40-column switch status on bit 7; 1 = 40-column display = switch down
\$C061	(⇧) status on bit 7; 1 = pressed (also game input switch 0)
\$C062	(⇩) status on bit 7; 1 = pressed

Game input switches: see Chapter 9.

**Table 4-1—Continued. Keyboard Input Characteristics**

	<b>Monitor Firmware Routines</b>		
	<b>Location</b>	<b>Name</b>	<b>Description</b>
GETLN, GETLN1, and RDKEY: see Chapter 3.	\$FD6A	GETLN	Gets an input line with prompt.
	\$FD67	GETLNZ	Gets an input line with preceding carriage return.
	\$FD6F	GETLN1	Gets an input line, but with no preceding prompt.
	\$FD1B	KEYIN	The keyboard input subroutine
	\$FD35	RDCHAR	Gets an input character or escape code.
	\$FD0C	RDKEY	The standard character input subroutine
<b>Use of Other Pages</b>			
	Page 2	The standard character string input buffer (see GETLN description)	

---

### 4.1.1 Reading the Keyboard

The keyboard encoder and ROM generate all 128 ASCII codes, so all the special character codes in the ASCII character set are available from the keyboard. Machine-language programs obtain character codes from the keyboard by using RDKEY, which reads a byte from the keyboard-data location shown in Table 4-1.

Here is how reading the keyboard is done:

1. To see if a key has been pressed, test bit 7 at address \$C000.
2. When that bit goes to a 1, the low-order seven bits are the character.
3. Clear the high bit at \$C000 by reading or writing anything to address \$C010.

\$C010 has another function: its high bit is a 1 while a key is pressed (except the Apple keys, **CONTROL**, **SHIFT**, **CAPS-LOCK**, and **RESET**). Bit 7 at this location is therefore called **any-key-down**.

Any time you read the any-key-down bit at \$C010, you also clear the keyboard strobe bit at \$C000. If your program needs to read both the flag and the strobe, it must read the strobe bit first.

After the keyboard strobe has been cleared, it remains low until another key is pressed.

For a description of how the keyboard strobe works, refer to Appendix E.



Table 4-2 shows the ASCII codes for the keys on the Apple IIc keyboard. If the strobe bit is set, add \$80 to these values.

**Table 4-2. Keys and ASCII Codes**

Key	Normal	Char	Control	Char	Shift	Char	Both	Char
<b>DELETE</b>	7F	DEL	7F	DEL	7F	DEL	7F	DEL
<b>←</b>	08	BS	08	BS	08	BS	08	BS
<b>TAB</b>	09	HT	09	HT	09	HT	09	HT
<b>␣</b>	0A	LF	0A	LF	0A	LF	0A	LF
<b>␣</b>	0B	VT	0B	VT	0B	VT	0B	VT
<b>RETURN</b>	0D	CR	0D	CR	0D	CR	0D	CR
<b>→</b>	15	NAK	15	NAK	15	NAK	15	NAK
<b>ESC</b>	1B	ESC	1B	ESC	1B	ESC	1B	ESC
SPACE	20	SP	20	SP	20	SP	20	SP
"	27	'	27	'	22	"	22	"
, <	2C	,	2C	,	3C	<	3C	<
_	2D	-	1F	US	5F	_	1F	US
. >	2E	.	2E	.	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0 )	30	0	30	0	29	)	29	)
1 !	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	^	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	*	2A	*
9 (	39	9	39	9	28	(	28	(
::	3B	;	3B	:	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[	5B	[	1B	ESC	7B	[	1B	ESC
\	5C	\	1C	FS	7C	\	1C	FS
]	5D	]	1D	GS	7D	]	1D	GS
~	60	~	60	~	7E	~	7E	~

Codes are shown here in hexadecimal; to find the decimal equivalents, refer to section H.6.



Table 4-2—Continued. Keys and ASCII Codes

Key	Normal	Char	Control	Char	Shift	Char	Both	Char
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

Keystrokes can also generate interrupts: see Appendix E.

This restarting process is called the **reset routine**, and is described in Chapter 2.

For information on how to have programs interpret keystrokes in a standard way, refer to the *Apple II Design Guidelines* listed in the Bibliography.

There are several special-function keys that do not generate ASCII codes. For example, you cannot read **CONTROL**, **SHIFT**, and **CAPS-LOCK** directly, but pressing one of these keys alters the character codes produced by the other keys. Programs can also read the status of  and  when checking keyboard input, and, if one or both of them is pressed, branch to a special routine, such as a help program.

Another key that doesn't generate a code is **RESET**, located at the upper-left corner of the keyboard; it is connected directly to the Apple IIc's circuits. Pressing **RESET** with **CONTROL** depressed normally causes the system to stop whatever program it's running and restart itself.

---

### **4.1.2 Monitor Firmware Support**

Chapter 3 describes the three standard Monitor input routines serving the keyboard: GETLN, READKEY and KEYIN. This section discusses the three other Monitor routines available.

---

#### **GETLNZ**

GETLNZ (at address \$FD67) is an alternate entry point for GETLN that sends a carriage return to the standard output, then continues into GETLN.

---

#### **GETLN1**

GETLN1 (at address \$FD6F) is an alternate entry point for GETLN that does not issue a prompt before it accepts the input line. However, if the user cancels the input line with too many backspaces or with **CONTROL-X**, then GETLN1 issues the prompt at location \$33 when it gets another line.

---

#### **RDCHAR**

RDCHAR (at address \$FD35) is a subroutine that gets characters from the standard input subroutine, and also interprets the escape codes listed in Chapter 3.

If the enhanced video firmware is active, **(-)((CONTROL-U))** causes a character to be read from the screen location and returned.

## 4.2 Speaker Output

Electrical specifications of the speaker circuit appear in Chapter 11.

The Apple IIc has a speaker mounted toward the front of the bottom plate. The speaker is connected to a soft switch that toggles; it has two states, off and on, and it changes from one to the other each time it is accessed. Table 4-3 describes the speaker output characteristics.

*Table 4-3. Speaker Output Characteristics*

<b>Port Number</b>	None	
<b>Commands</b>	Some programs sound the speaker in response to CONTROL-G.	
<b>Initial Characteristics</b>	Reset routine sounds the speaker.	
<b>Hardware Location</b>	<b>Description</b>	
SC030	Toggle speaker (read only)	
<b>Monitor Firmware Location</b>	<b>Routines Name</b>	<b>Description</b>
\$FBDD	BELL1	Sends a beep to the speaker.
\$FF3A	BELL	Sends CONTROL-G to the current output

### 4.2.1 Using the Speaker

If you switch the speaker once, it emits a click; to make longer sounds, you access the speaker repeatedly. You should always use a read operation to toggle the speaker. If you write to this soft switch, it switches twice in rapid succession. The resulting pulse is so short that the speaker doesn't have time to respond; it doesn't make a sound.

The switch for the speaker uses memory location SC030. You can make various tones and buzzes with the speaker by using combinations of timing loops in your program.

See Chapter 3.

---

### **4.2.2 Monitor Firmware Support**

The Monitor supports the speaker with one simple routine, BELL1. A related routine, BELL, supports the current output device—the one that CSW points to.

---

#### ***BELL1***

BELL1 (at address \$FDBB) makes a beep through the speaker by generating a 1 kHz tone in the Apple IIc's speaker for 0.1 second. This routine scrambles the A and X registers.

---

#### ***BELL***

The Monitor routine BELL (at location \$FF3A) writes a bell control character (ASCII CONTROL-G) to the current output device. This routine leaves the accumulator holding \$87.



[illegible]

# *Video Display Output*

NTSC stands for National Television Standards Committee, a group that formulates broadcast and reception guidelines used by the USA and several other countries.

The primary output device of the Apple IIc is the video display. You can use any ordinary video monitor, either color or monochrome, to display video information from the Apple IIc. An ordinary monitor is one that accepts composite video compatible with the standard set by the NTSC. If you use Apple IIc color graphics with, for example, a black-and-white monitor, the display will appear as black, white, and three shades of gray.

If you are only using 40-column text and graphics modes, you can use a television set for your video display. If the TV set has an input connector for composite video, you can connect it directly to your Apple IIc; if it does not, you must attach an RF video modulator between the Apple IIc and the television set.

**Note:** The Apple IIc can produce an 80-column text display. However, if you use an ordinary color or black-and-white television set, 80-column text will be too blurry to read. For a clear 80-column display, you must use a high-resolution video monitor with a bandwidth of 14 MHz or greater.

Table 5-1 is a summary of the video output port's characteristics and a guide to other information in this chapter.

**Table 5-1.** *Guide to the Information in This Chapter*

<b>Port Number</b>	Output port 3
<b>Commands</b>	Figure 5-3
<b>Initial Characteristics</b>	Figure 5-2 <b>Note:</b> If a program is to use the enhanced video firmware, it should turn it on and then immediately check the 80/40 switch. If the switch is in the 40 position, the program should issue a CONTROL-Q.
<b>Hardware Locations</b>	See Table 5-7
<b>Monitor Firmware Routines</b>	See Table 5-8
<b>I/O Firmware Entry Points</b>	See Table 5-9

## 5.1 Specifications

Table 5-2 summarizes the specifications for the video display, and provides a further guide to other information in this chapter.

*Table 5-2. Video Display Specifications*

Display modes	40-column text; map: Figure 5-5 80-column text; map: Figure 5-6  Low-resolution color graphics; map: Figure 5-7  High-resolution color graphics; map: Figure 5-8  Double-high-resolution color graphics; map: Figure 5-9
Text capacity	24 lines by 80 columns (character positions)
Character set	96 ASCII characters (uppercase and lowercase)
Display formats	Normal, Inverse, Flashing, MouseText (Table 5-3)
Low-resolution graphics	16 colors (Table 5-4) 40 horizontal by 48 vertical; map: Figure 5-7
High-resolution graphics	6 colors (Table 5-5) 140 horizontal by 192 vertical (restricted);  black-and-white: 280 horizontal by 192 vertical; map: Figure 5-8
Double-high-resolution graphics	16 colors (Table 5-6) 140 horizontal by 192 vertical (no restrictions);  black-and-white: 560 horizontal by 192 vertical; map: Figure 5-9

The video signal produced by the Apple IIc is NTSC-compatible composite color video. It is available at two places: the RCA-type phono jack on the back of the Apple IIc, and the 15-pin D-type connector on the back panel. Use the RCA-type phono jack to connect a video monitor or the DB-15 connector for an external video modulator; use the 15-pin connector to attach video expansion hardware (section 11.9.5).

Either of the two text modes can display all 96 ASCII characters: uppercase and lowercase letters, numbers, and symbols.



## 5.2 Text Modes

The text characters displayed include the uppercase and lowercase letters, the ten digits, punctuation marks, and special characters. Each character is displayed in an area of the screen that is seven dots wide by eight dots high. The characters are formed by a dot matrix five dots wide (with a few exceptions, such as underscore), leaving two blank columns of dots between characters in a row. Except for lowercase letters with descenders, the characters are only seven dots high, leaving one blank line of dots between rows of characters.

The normal display has white (or other monochrome color) dots on a dark background. Characters can also be displayed as black dots on a white background; this is called inverse format.

---

### 5.2.1 Text Character Sets

The Apple IIc can display either of two text character sets: the primary set and an alternate set (Table 5-3). The forms of the characters in the two sets are actually the same, but the available display formats are different. The display formats are

- **normal**, with white dots on a black screen
- **inverse**, with black dots on a white screen
- **flashing**, alternating between normal and inverse.

With the primary character set, the Apple IIc can display uppercase characters in all three formats: normal, inverse, and flashing. Lowercase letters can only be displayed in normal format. The primary character set is compatible with most software written for the Apple II and II Plus, which can display text in flashing format but don't have lowercase characters.

The alternate character set sacrifices the flashing format for a complete inverse format. With the alternate character set, the Apple IIc can display uppercase letters, lowercase letters, numbers, and special characters in either normal format or inverse format. It can also display MouseText.

MouseText: see section 5.2.2.

You select the character set by means of the alternate-text soft switch, described in section 5.6. Table 5-3 shows the character codes in decimal and hexadecimal for the Apple IIc primary and alternate character sets in normal, inverse, and flashing formats.

Each character on the screen is stored as one byte of display data. The low-order six bits make up the ASCII code of the character being displayed. The remaining two (high-order) bits select format and the group within ASCII (section 3.3.6).

*Table 5-3. The Display Character Sets*

To identify particular characters and values, refer to Table 4-2.

Hex Values	Primary Character Set		Alternate Character Set	
	Character Type	Format	Character Type	Format
\$00-\$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
\$20-\$3F	Special characters	Inverse	Special characters	Inverse
\$40-\$5F	Uppercase letters	Flashing	MouseText (section 5.2.2)	
\$60-\$7F	Special characters	Flashing	Lowercase letters	Inverse
\$80-\$9F	Uppercase letters	Normal	Uppercase letters	Normal
\$A0-\$BF	Special characters	Normal	Special characters	Normal
\$C0-\$DF	Uppercase letters	Normal	Uppercase letters	Normal
\$E0-\$FF	Lowercase letters	Normal	Lowercase letters	Normal

### 5.2.2 MouseText

The character-generator ROM can display 32 graphics characters called **MouseText** in place of the inverse uppercase series from \$40 through \$5F. These graphics are especially convenient to use with a mouse, since they can be generated by character codes instead of groups of high-resolution byte values, and they can be moved around quickly. To use MouseText characters, do the following:

1. Turn on the enhanced video firmware: issue PR#3 or ESC 8 or ESC 4.
2. Set inverse mode: use the INVERSE command or put \$3F in location \$32, or print CONTROL-O.
3. Turn on the MouseText feature: PRINT CHR\$(27); or pass \$1B to COUT in the accumulator.
4. Print the uppercase letter (or other ASCII character in the range \$40-\$5F: @ [ \ ] ^ or \_ ) that corresponds to the MouseText character you want, using COUT1.

5. Turn off the MouseText feature: `PRINT CHR$(24);` or pass \$18 to COUT1 in the accumulator.
6. Set normal mode: use the `NORMAL` command or put \$FF in location \$32, or print a `CONTROL-N`.

Here is a sample Applesoft program that prints all the MouseText characters:

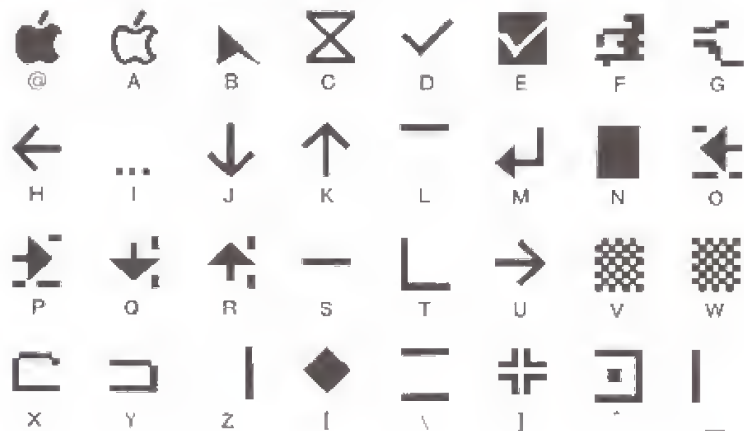
```

10 D$=CHR$(4)
20 PRINT PRINT D$;"PR#3"
30 INVERSE
40 PRINT
CHR$(27);"@ABCDEFGHIJKLMN O PQRSTUVWXYZ[\]^_";
50 PRINT CHR$(24);
60 NORMAL

```

Simulated MouseText characters and their corresponding ASCII characters are shown in Figure 5-1.

*Figure 5-1. MouseText Characters*



---

### ***5.2.3 40-Column Versus 80-Column Text***

The Apple IIc has two modes of text display: 40-column and 80-column. The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare the two displays in Figure 5-2. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

Figure 5-2. 40-Column and 80-Column Text Display  
(With Alternate Character Set)

```

10 LIST 0,100
10 REM APPLESOFT CHARACTER DEMO
20 TEXT HOME
30 PRINT PRINT "Applesoft Char
acter Demo"
40 PRINT PRINT "Which character
set?"
50 PRINT INPUT "Primary (P) or
Alternate (A)?" A$
60 IF LEN(A$) < 1 THEN 50
70 LET A$ = LEFT$(A$,1)
80 IF A$ = "P" THEN POKE 49166,
0
90 IF A$ = "A" THEN POKE 49167,
0
100 PRINT PRINT "Printing up
the same line first"
110 PRINT "in NORMAL then INVERSE
then FLASH" PRINT

```

```

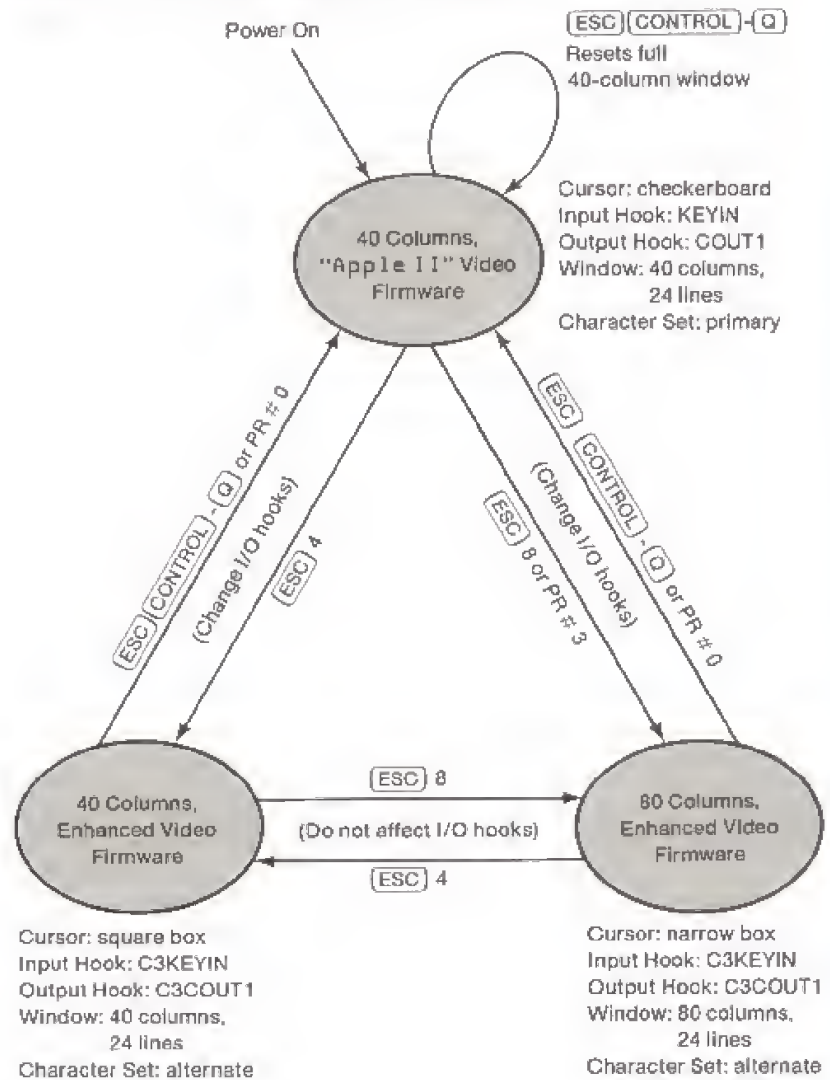
1.87
10 REM APPLESOFT CHARACTER DEMO
20 TEXT HOME
30 PRINT PRINT "Applesoft Character Demo"
40 PRINT PRINT "Which character set?"
50 PRINT INPUT "Primary (P) or Alternate (A)?" A$
60 IF LEN(A$) < 1 THEN 50
70 LET A$ = LEFT$(A$,1)
80 IF A$ = "P" THEN POKE 49166,0
90 IF A$ = "A" THEN POKE 49167,0
100 PRINT PRINT "Printing the same line first"
110 PRINT "in NORMAL then INVERSE then FLASH" PRINT
120 GOSUB 1000
130 GOSUB 1000
140 GOSUB 1000
150 PRINT PRINT "Press any key to repeat"
160 GOTO 10
170 PRINT "SAMPLE TEXT: Now is the time-- 12:47"

```



Figure 5-3 illustrates the methods of switching text display modes, and the characteristics of each.

Figure 5-3. Text Mode Characteristics and Switching



## 5.3 Graphics Modes

The Apple IIc can produce video graphics in any of three different modes. Each graphics mode treats the screen as a rectangular array of spots. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes in these arrays; this section describes the way the resulting graphics data are stored in the Apple IIc's memory.

### 5.3.1 Low-Resolution Graphics

In the low-resolution graphics mode, the Apple IIc displays an array of 48 rows by 40 columns of colored blocks. Each block can be any one of sixteen colors, including black and white. On a black-and-white monitor or television set, these colors appear as black, white, and two shades of gray. There are no blank dots between blocks; adjacent blocks of the same color merge to make a larger shape.

Data for the low-resolution graphics display is stored in the same part of memory as the data for the 40-column text display. Each byte contains data for two low-resolution graphics blocks. The two blocks are displayed one atop the other in a display space the same size as a 40-column text character, seven dots wide by eight dots high.

Half a byte—four bits, or one nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the screen. The colors and their corresponding nibble values are shown in Table 5-4. In each byte, the low-order nibble sets the color for the top block of the pair, and the high-order nibble sets the color for the bottom block. Thus, a byte containing the hexadecimal value \$D8 produces a brown block atop a yellow block on the screen.

Colors may vary, depending upon the controls on the monitor or television set.

*Table 5-4. Low-Resolution Graphics Colors*

Nibble Value			Nibble Value		
Decimal	Hex	Color	Decimal	Hex	Color
0	\$0	Black	8	\$8	Brown
1	\$1	Magenta	9	\$9	Orange
2	\$2	Dark Blue	10	\$A	Gray 2
3	\$3	Purple	11	\$B	Pink
4	\$4	Dark Green	12	\$C	Light Green
5	\$5	Gray 1	13	\$D	Yellow
6	\$6	Medium Blue	14	\$E	Aquamarine
7	\$7	Light Blue	15	\$F	White

As explained in section 5.5, the text display and the low-resolution graphics display use the same area in memory. Most programs that generate text and graphics clear this part of memory when they change display modes, but it is possible to store data as text and display it as graphics, or vice versa. All you have to do is change the mode switch, described in section 5.6, without changing the display data. This usually produces meaningless jumbles on the display, but some programs have used this technique to good advantage for producing complex low-resolution graphics displays quickly.

### **5.3.2 High-Resolution Graphics**

In the high-resolution graphics mode, the Apple IIc displays an array of colored dots in 192 rows and 280 columns. The colors available are black, white, purple, green, orange, and blue, although the colors of the individual dots are limited, as described below. Adjacent dots of the same color merge to form a larger colored area.

Data for the high-resolution graphics displays is stored in either of two 8192-byte areas in memory. These areas are called high-resolution Page 1 and Page 2; think of them as buffers where you can put data to be displayed. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes to display; this section describes the way the resulting graphics data are stored in the Apple IIc's memory.

The Apple IIc high-resolution graphics display is bit-mapped: each dot on the screen corresponds to a bit in the Apple IIc's memory. The seven low-order bits of each display byte control

a row of seven adjacent dots on the screen, and forty adjacent bytes in memory control a row of 280 (7 times 40) dots. The least significant bit of each byte is displayed as the leftmost dot in a row of seven, followed by the second-least significant bit, and so on, as shown in Figure 5-4. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described below.

On a black-and-white monitor, there is a simple correspondence between bits in memory and dots on the screen. A dot is white if the bit controlling it is on (1), and the dot is black if the bit is off (0). On a black-and-white television set, pairs of dots blur together; alternating black and white dots merge to a continuous grey.

On an NTSC color monitor or a color television set, a dot whose controlling bit is off (0) is black. If the bit is on, the dot will be white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte. Call the left-most column of dots column zero, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control them are on, dots in even-numbered columns, 0, 2, 4, and so forth, are purple, and dots in odd-numbered columns are green—but only if the dots on either side are black. If two adjacent dots are both on, they are both white.

You select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored dots controlled by a byte with the high-order bit on are either blue or orange: the dots in even-numbered columns are blue, and the dots in odd-numbered columns are orange—again, only if the dots on either side are black. Within each horizontal line of seven dots controlled by a single byte, you can have black, white, and one pair of colors. To change the color of any dot to one of the other pair of colors, you must change the high-order bit of its byte, which affects the colors of all seven dots controlled by the byte.

In other words, high-resolution graphics displayed on a color monitor or television set are made up of colored dots, according to the following rules:

- Dots in even columns can be black, purple, or blue.
- Dots in odd columns can be black, green, or orange.
- If adjacent dots in a row are both on, they are both white.
- The colors in each row of seven dots controlled by a single byte are either purple and green, or blue and orange, depending on whether the high-order bit is off (0) or on (1).

These rules are summarized in Table 5-5. The blacks and whites are numbered to remind you that the high-order bit is different.

*Table 5-5. High-Resolution Graphics Colors*

Bits 0-6	Bit 7 Off	Bit 7 On
Adjacent columns off	Black 1	Black 2
Even columns on	Purple	Blue
Odd columns on	Green	Orange
Adjacent columns on	White 1	White 2

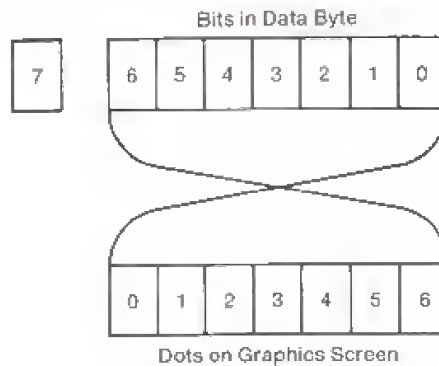
Colors may vary, depending on the adjustment of the monitor or television set.

For more details about the way the Apple IIc produces color on a TV set, see Chapter 11. For a table of reversed bit patterns, refer to Appendix H. For information about the way NTSC color television works, see the magazine articles listed in the Bibliography.

The peculiar behavior of the high-resolution colors reflects the way NTSC color television works. The dots that make up the Apple IIc video signal are spaced to coincide with the frequency of the color subcarrier used in the NTSC system. Alternating on and off dots at this spacing cause a color monitor or TV set to produce color, but two or more on dots together do not.



Figure 5-4. High-Resolution Display Bits



### 5.3.3 Double-High-Resolution Graphics

Double-high-resolution graphics is a bit-mapping of the low-order seven bits of the bytes in the two high-resolution graphics pages. The bytes in the main-memory and auxiliary-memory pages are interleaved in exactly the same manner as the characters in 80-column text: of each pair of identical addresses, the auxiliary-memory byte is displayed first, and the main-memory byte is displayed second. Horizontal resolution is 560 dots when displayed on a monochrome monitor.

Unlike high-resolution color (section 5.3.2), double-high-resolution color has no restrictions on which colors can be adjacent. Color is determined by any four adjacent dots along a line. Think of a 4-dot-wide window moving across the screen: at any given time, the color displayed will correspond to the four-bit value from Table 5-6 that corresponds to the window's position (Figure 5-9). Effective horizontal resolution with color is 140 (560 divided by four).

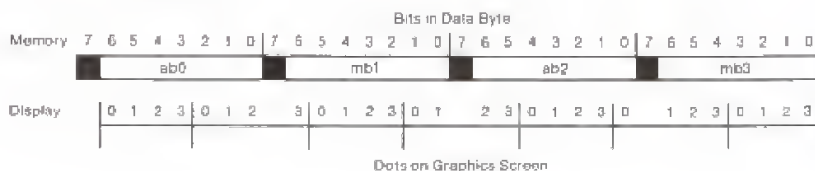
To use the table, divide the column number by 4, and use the remainder to find the correct column: *ab0* is a byte residing in auxiliary memory corresponding to a remainder of 0 (byte 0, 4, 8, and so on), *mb1* is a byte residing in main memory corresponding to a remainder of 1 (byte 1, 2, 9, and so on), and similarly for *ab2* and *mb3*.

Colors may vary, depending upon the controls on the monitor or television set.

**Table 5-6. Double-High-Resolution Graphics Colors**

Color	ab0	mb1	ab2	mb3	Repeated Bit Pattern
Black	\$00	\$00	\$00	\$00	0000
Magenta	\$08	\$11	\$22	\$44	0001
Brown	\$44	\$08	\$11	\$22	0010
Orange	\$4C	\$19	\$33	\$66	0011
Dark Green	\$22	\$44	\$08	\$11	0100
Gray 1	\$2A	\$55	\$2A	\$455	0101
Green	\$66	\$4C	\$19	\$33	0110
Yellow	\$6E	\$5D	\$3B	\$77	0111
Dark Blue	\$11	\$22	\$44	\$08	1000
Purple	\$19	\$33	\$66	\$4C	1001
Gray 2	\$55	\$2A	\$55	\$2A	1010
Pink	\$5D	\$3B	\$77	\$6E	1011
Medium Blue	\$33	\$66	\$4C	\$19	1100
Light Blue	\$3B	\$77	\$6E	\$5D	1101
Aqua	\$77	\$6E	\$5D	\$3B	1110
White	\$7F	\$7F	\$7F	\$7F	1111

■ — Unused bit



## 5.4 Mixed-Mode Displays

Any of the graphics displays can have four lines of text, either 40-column or 80-column, at the bottom of the screen. Graphics displays with text at the bottom are called **mixed-mode** displays. To use them, the TEXT switch must be off (read \$C050) and the MIXED switch on (read \$C053).

**Note:** You cannot display 40-column text with double-high-resolution graphics.

To determine what appears where in mixed-mode displays, refer to the upper five-sixths (down to the heavy horizontal line) in the appropriate graphics display (Figures 5-7 to 5-9); then refer to the bottom sixth of the appropriate text display (Figure 5-5 or 5-6).

## 5.5 Display Pages

The Apple IIc generates its video displays using data stored in specific areas in memory. These areas, called **display pages**, serve as buffers where your programs can put data to be displayed. Each byte in a display buffer controls an object at a certain location on the display: a character, a colored block, or a group of adjacent dots.

The 40-column-text and low-resolution-graphics modes use two display pages of 1024 bytes each. These are called Text Page 1 and Text Page 2, and they are located at \$400-\$7FF and \$800-\$BFF in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch displays instantly. Either page can be displayed as 40-column text, low-resolution graphics, or mixed-mode (four rows of text at the bottom of a graphics display).

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages when the enhanced video firmware is active. The 80-column text display uses a combination page made up of Text Page 1 in main memory plus another page in auxiliary memory. This additional memory is *not* the same as Text Page 2—in fact, it is Text Page 1X, and it occupies the same address space as Text Page 1 (see Figure 2-11). The built-in firmware I/O routines

described in Chapter 3 take care of this extra addressing automatically; that is one reason to use these routines for all normal text output.

**Note:** The built-in video firmware always displays Page 1 text. You cannot write text to Page 2 unless you do it yourself.

The high-resolution graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and low-resolution graphics modes each byte controls a display area seven dots wide by eight dots high. In high-resolution graphics mode each byte controls an area seven dots wide by one dot high. Thus, a high-resolution display requires eight times as much data storage, as shown in Table 5-7.

The double-high-resolution graphics mode interleaves the two high-resolution Pages (1 and 1X) in exactly the same way as 80-column text mode interleaves the text pages: column 0 and all subsequent even-numbered columns come from the auxiliary page; column 1 and all subsequent odd-numbered columns come from the main page.

*Table 5-7. Video Display Page Locations. (1) This is not supported by firmware; for instructions on how to switch pages, refer to section 5.6. (2) See section 5.3.3.*

Display Mode	Display Page	Lowest Address		Highest Address	
40-column text, low-resolution graphics	1	\$400	1024	\$7FF	2047
	2 (1)	\$800	2048	\$BFF	3071
80-column text	1	\$400	1024	\$7FF	2047
	2 (1)	\$800	2048	\$BFF	3071
high-resolution graphics	1	\$2000	8192	\$3FFF	16383
	2	\$4000	16384	\$5FFF	24575
double-high- resolution graphics	1 (2)	\$2000	8192	\$3FFF	16383
	2 (2)	\$4000	16384	\$5FFF	16383

## 5.6 Display Mode Switching

Table 5-8 shows the reserved locations for the soft switches that control the different display modes. The left column of the table, labelled Action, indicates what to do to activate or read a switch setting: R means read the location, W means write anything to the location, R/W means read or write, and R7 means read the location and then check bit 7.

Table 5-9 lists the display modes that the firmware can set up automatically. In the 40-column modes, the contents of the standard I/O hooks KSW and CSW (Chapter 3) determine whether the enhanced video firmware features are available or not. The firmware also takes care of setting or clearing ALTCHAR.

Table 5-10 lists other display modes available but not supported by firmware. For modes that display Page 2 with the 80COL switch on, your program may have to turn 80STORE off after the firmware has turned it on.

Double-low-resolution shows on the display screen when HIRES is off and both 80COL and DHIRES are on. It is the low-resolution graphics equivalent of 80-column text, and it uses the same map (Figure 5-6).

The IOUDIS switch must be on for locations \$C05E and \$C05F to change DHIRES. The firmware in fact leaves it on—and your program should, too—unless it wants to use locations \$C05E and \$C05F to change mouse values (Chapter 9).



**Table 5-8. Display Soft Switches.** (1) The firmware normally leaves *LOUDIS* on. See also note 2. (2) Reading or writing any address in the range *\$C070-\$C07F* also triggers the paddle timer and resets *VBLINT* (Chapter 9).

Action	Hex	Name	Function	Note
W	\$C00E	ALTCHAR	Off: display text using primary character set	
W	\$C00F	ALTCHAR	On: display text using alternate character set	
R7	\$C01E	RDALTCHAR	Read ALTCHAR switch (1 = on)	
W	\$C00C	80COL	Off: display 40 columns	
W	\$C00D	80COL	On: display 80 columns	
R7	\$C01F	RD80COL	Read 80COL switch (1 = on)	
W	\$C000	80STORE	Off: cause PAGE2 on to select auxiliary RAM	
W	\$C001	80STORE	On: allow PAGE2 to switch main RAM areas	
R7	\$C018	RD80STORE	Read 80STORE switch (1 = on)	
R/W	\$C054	PAGE2	Off: select page 1	
R/W	\$C055	PAGE2	On: select page 1X (80STORE on) or 2	
R7	\$C01C	RDPAGE2	Read PAGE2 switch (1 = on)	
R/W	\$C050	TEXT	Off: display graphics or (if MIXED on) mixed	
R/W	\$C051	TEXT	On: display text	
R7	\$C01A	RDTEXT	Read TEXT switch (1 = on)	
R/W	\$C053	MIXED	Off: display only text or only graphics	
R/W	\$C054	MIXED	On: (if TEXT off) display text and graphics	
R7	\$C01B	RDMIXED	Read MIXED switch (1 = on)	
R/W	\$C057	HIRES	Off: (if TEXT off) display low-resolution graphics	
R/W	\$C058	HIRES	On: (if TEXT off) display high-resolution or (if DHIRES on) double-high-resolution graphics	
R7	\$C01D	RDHIRES	Read HIRES switch (1 = on)	

*Table 5-8—Continued. Display Soft Switches*

Action	Hex	Name	Function	Note
W	\$C07E	IOUDIS	On: disable IOU access for addresses \$C058 to \$C05F; enable access to DHIRES switch	(1)
W	\$C07F	IOUDIS	Off: enable IOU access for addresses \$C058 to \$C05F; disable access to DHIRES switch	(1)
R7	\$C07E	RДИОUDIS	Read IOUDIS switch (1 = off)	(2)
R/W	\$C05E	DHIRES	On: (if IOUDIS on) turn on double-high-resolution	
R/W	\$C05F	DHIRES	Off: (if IOUDIS on) turn off double-high-resolution	
R7	\$C07F	RDDHIRES	Read DHIRES switch (1 = on)	(2)

**Table 5-9. Display Modes Supported by Firmware (Including Applesoft).** (1) 80STORE is set by the firmware when 80COL is turned on.

Display Col/Res	Type	Page	Switches						
			80COL	80STORE	PAGE2	TEXT	MIXED	HIRES	DHIRES
40-column	text	1	off		off	on	off	off	off
80-column	text	1	on	(1)		on			
low-res	graphics	1	off		off	off	off	off	off
40/low	mixed	1	off		off	off	on	off	
80/low	mixed	1	on	(1)	off	off	on	off	off
hi-res	graphics	1	off		off	off	off	on	
hi-res	graphics	2	off		on	off	off	on	
40/high	mixed	1	off		off	off	on	on	
80/high	mixed	1	on	(1)	off	off	on	on	off

**Table 5-10. Other Display Modes.** (1) 80STORE is set by the firmware when 80COL is turned on, and must be turned off to use the second 80-column or double-high-resolution page. This means that you cannot use firmware routines such as COUT when displaying Page 2 modes not supported by firmware.

Display Col/Res	Type	Page	Switches						
			80COL	80STORE	PAGE2	TEXT	MIXED	HIRES	DHIRES
40-column	text	2	off		on	on			
80-column	2	on	off	on	on				
low-res	graphics	2	off		on	off	off	off	
40/low	mixed	2	off		on	off	on	off	
80/low	mixed	2	on	off	on	off	on	off	off
dbl-low	graphics	1	on	(1)	off	off	off	off	on
dbl-low	graphics	2	on	off	on	off	off	off	on
80/dbl-low	mixed	1	on	(1)	off	off	on	off	on
80/dbl-low	mixed	2	on	off	on	off	on	off	on
40/high	mixed	2	off		on	off	on	on	
80/high	mixed	2	on	off	on	off	on	on	off
dbl-high	graphics	1	on	(1)	off	off	off	on	on
dbl-high	graphics	2	on	off	on	off	off	on	on
80/dbl-high	mixed	1	on	(1)	off	off	on	on	on
80/dbl-high	mixed	2	on	off	on	off	on	on	on

For example, to switch to mixed 80-column and double-high-resolution display Page 1, you can use these instructions:

STA	\$C00D	Turn on 80COL; firmware then turns on 80STORE
LDA	\$C054	Turn off PAGE2; you could also have done a STA
STA	\$C050	Turn off TEXT; that is, turn on graphics mode
STA	\$C053	Turn on MIXED; it works now that TEXT is off
STA	\$C057	Turn on HIRES; it works now that TEXT is off
STA	\$C07E	Make sure IOUDIS is on so you can access DHIRES
LDA	\$C05E	Turn on DHIRES; it works now that

---

## 5.7 Display Page Maps

You should never have to store directly into display memory. Most high-level languages enable you to write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you should be able to use the display features of the built-in I/O firmware.

---

### Warning

*Never call any firmware with 80COL on or with 80STORE and PAGE2 both on. If you do, the firmware will not function properly. As a general rule, always leave PAGE2 off.*

---

The display memory maps are shown in Figures 5-5 through 5-9. All the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes (\$80 hexadecimal). For a full description of the way the Apple IIc handles its display memory, refer to section 11.9.2.

The high-resolution graphics display is stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of dots occupy the same space on the display as one row of characters. The subset consisting of all the first rows from the groups of eight is stored in the first 1024 bytes of the high-resolution display page. The subset consisting of all the second rows from the groups of eight is stored in the second 1024 bytes, and so on for a total of 8 times 1024, or 8192 bytes. In other words, each block of 1024 bytes in the high-resolution display page contains one row of dots out of every group of eight rows. The individual rows are stored in sets of three forty-byte rows, the same way as the text display.

The display maps show addresses only for each Page 1. To obtain addresses for text or low-resolution graphics Page 2, add 1024 (\$400); to obtain addresses for high-resolution Page 2, add 8192 (\$2000).

The 80-column display works a little differently. Half of the data is stored in the normal text Page 1 memory, and the other half is stored in the auxiliary memory text Page 1. The display circuitry fetches bytes from these two memory areas simultaneously and displays them sequentially: first the byte from the auxiliary memory, then the byte from the main memory. The main memory stores the characters in the odd columns of the display, and the auxiliary memory stores the characters in the even columns (starting with column 0 on the left).

For more details about the way the displays are generated, see Chapter 11.

To store display data in auxiliary memory, first turn on the 80STORE soft switch by writing to location \$C001. With 80STORE on, the page-select switch PAGE2 selects between the portion of the 80-column display stored in Page 1 of main



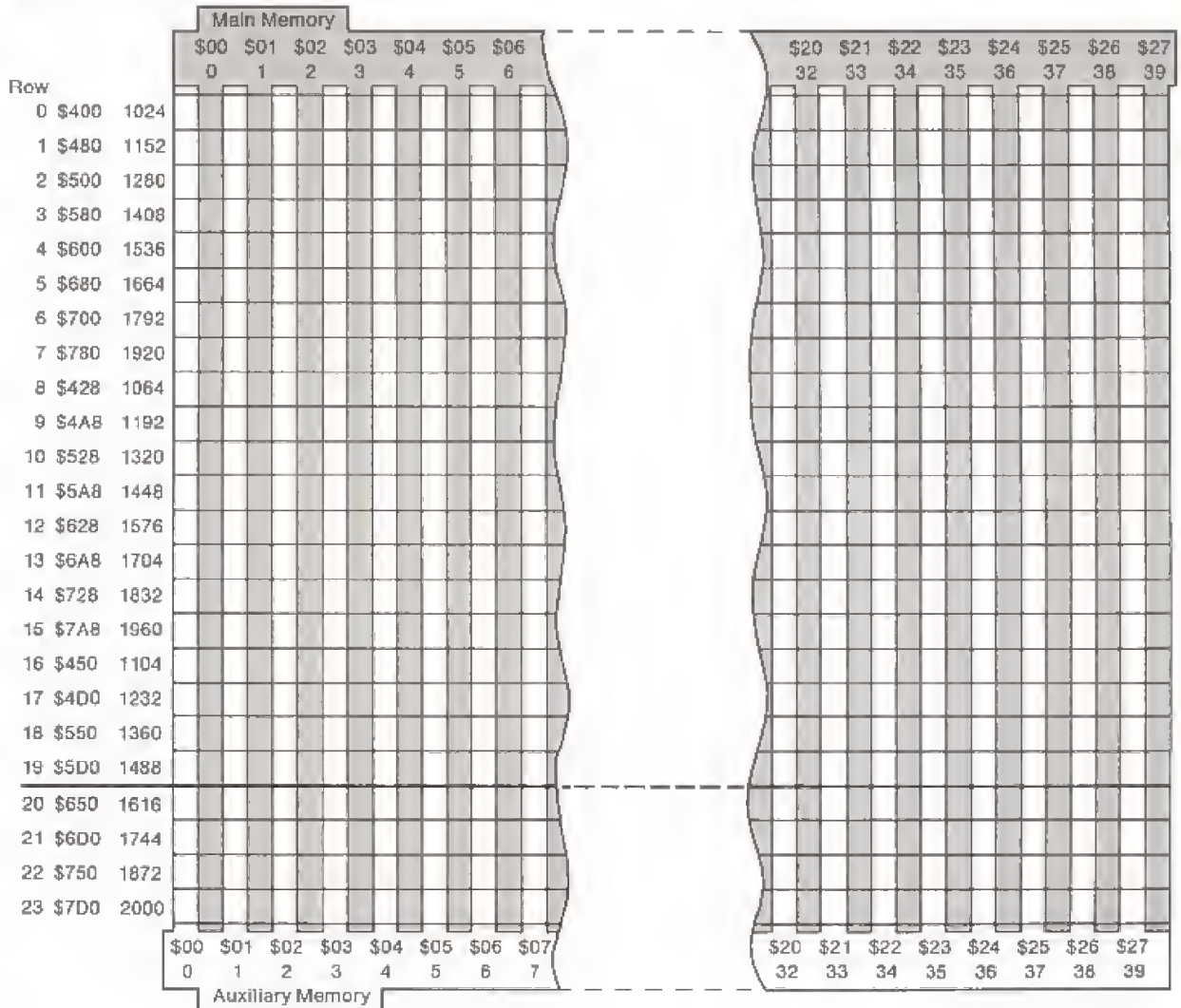
memory and the portion stored in the auxiliary memory. To select auxiliary memory, turn the PAGE2 soft switch on by reading or writing at location \$C055.

The double-high-resolution graphics display stores information in the same way as high-resolution graphics, except there is an auxiliary memory location as well as a main memory location corresponding to each address. The two sets of display information are interleaved in a manner similar to the interleaving of two 40-column displays to create an 80-column text display (Figure 5-9).

Figure 5-5. Map of 40-Column Text Display

			0 \$00	1 \$01	2 \$02	3 \$03	4 \$04	5 \$05	6 \$06	7 \$07	8 \$08	9 \$09	10 \$0A	11 \$0B	12 \$0C	13 \$0D	14 \$0E	15 \$0F	16 \$10	17 \$11	18 \$12	19 \$13	20 \$14	21 \$15	22 \$16	23 \$17	24 \$18	25 \$19	26 \$1A	27 \$1B	28 \$1C	29 \$1D	30 \$1E	31 \$1F	32 \$20	33 \$21	34 \$22	35 \$23	36 \$24	37 \$25	38 \$26	39 \$27	
Row			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	
0	\$400	1024																																									
1	\$480	1152																																									
2	\$500	1280																																									
3	\$580	1408																																									
4	\$600	1536																																									
5	\$680	1664																																									
6	\$700	1792																																									
7	\$780	1920																																									
8	\$428	1064																																									
9	\$4A8	1192																																									
10	\$528	1320																																									
11	\$5A8	1448																																									
12	\$628	1576																																									
13	\$6A8	1704																																									
14	\$728	1832																																									
15	\$7A8	1960																																									
16	\$450	1104																																									
17	\$4D0	1232																																									
18	\$550	1360																																									
19	\$5D0	1488																																									
20	\$650	1616																																									
21	\$6D0	1744																																									
22	\$750	1872																																									
23	\$7D0	2000																																									

*Figure 5-6. Map of 80-Column Text Display*

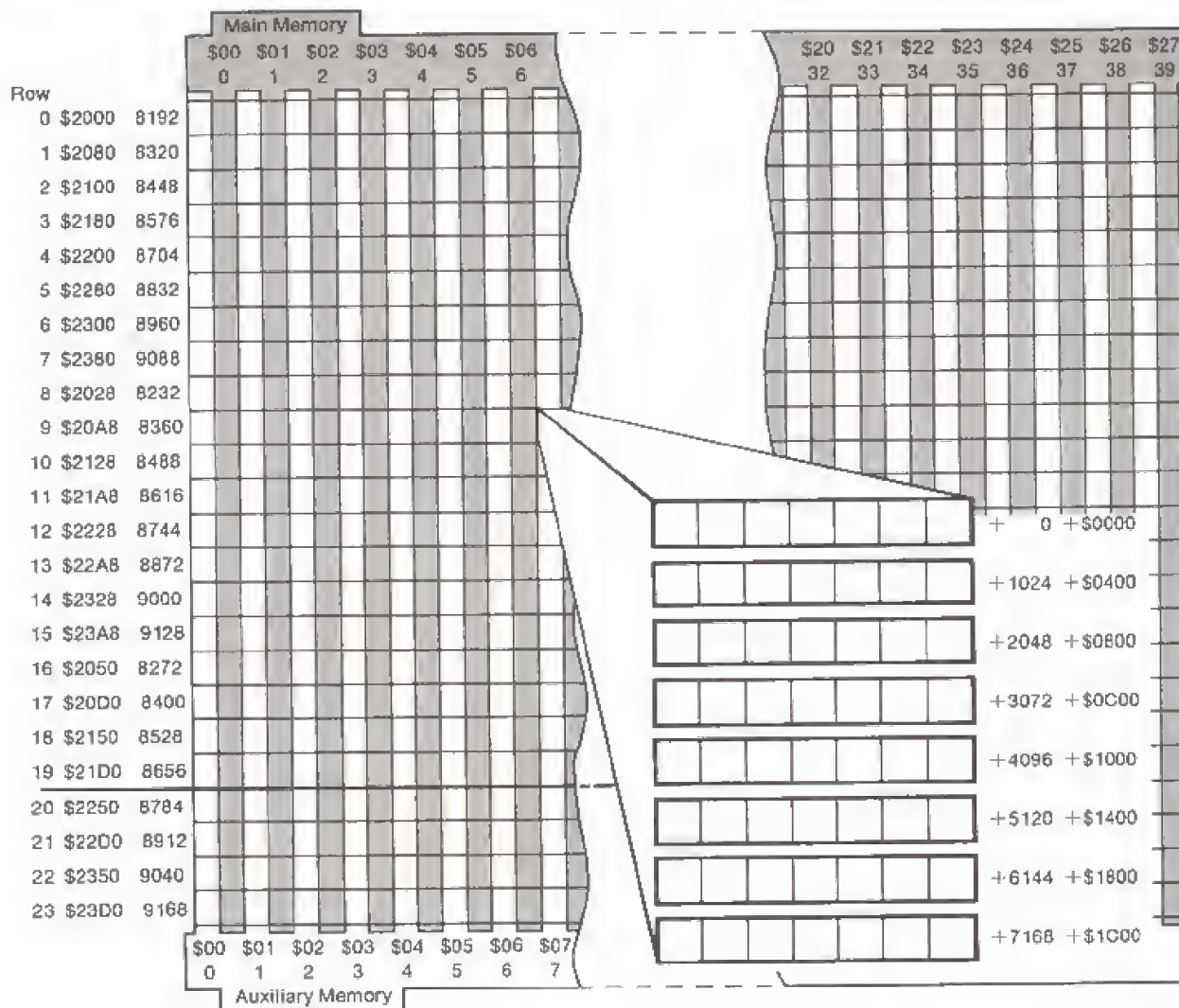




[illegible]



Figure 5-9. Map of Double-High-Resolution Graphics Display





## 5.8 Monitor Firmware Support

Table 5-9 summarizes the addresses and functions of the video display support routines the Monitor provides. Except for COUT and COUT1, which are explained in Chapter 3, these routines are described in the subsections that follow.

*Table 5-11. Monitor Firmware Routines*

Location	Name	Description
\$FC9C	CLREOL	Clears to end of line from current cursor position
\$FC9E	CLEOLZ	Clears to end of line using contents of Y register as cursor position
\$FC42	CLREOP	Clears to bottom of window
\$F832	CLRSCR	Clears the low-resolution screen
\$F836	CLRTOP	Clears top 40 lines of low-resolution screen
\$FDED	COUT	Calls output routine whose address is stored in CSW (normally COUT1, Chapter 3)
\$FDF0	COUT1	Displays a character on the screen (Chapter 3)
\$FD8E	CROUT	Generates a carriage return character
\$FD8B	CROUT1	Clears to end of line, then generates a carriage return character
\$F819	HLINE	Draws a horizontal line of blocks
\$FC58	HOME	Clears the window and puts cursor in upper-left corner of window

*Table 5-11—Continued. Monitor Firmware Routines*

Location	Name	Description
\$F800	PLOT	Plots a single low-resolution block on the screen
\$F94A	PRBL2	Sends 1 to 256 blank spaces to the output device whose address is in CSW
\$FDDA	PRBYTE	Prints a hexadecimal byte
\$FF2D	PRERR	Sends ERR and CONTROL-G to the output device whose output routine address is in CSW
\$FDE3	PRHEX	Prints 4 bits as a hexadecimal number
\$F941	PRTAX	Prints contents of A and X in hexadecimal
\$F871	SCRN	Reads color value of a low-resolution block on the screen
\$F864	SETCOL	Sets the color for plotting in low-resolution
\$FC24	VTABZ	Sets cursor vertical position (Setting CV at location \$25 does not change vertical position until a carriage return.)
\$F828	VLINE	Draws a vertical line of low-resolution blocks

---

#### **CLREOL**

CLREOL clears a text line from the cursor position to the right edge of the window. This routine destroys the contents of A and Y.

---

#### **CLEOLZ**

CLEOLZ clears a text line to the right edge of the window, starting at the location given by base address BASL indexed by the contents of the Y register. This routine destroys the contents of A and Y.

---

#### **CLREOP**

CLREOP clears the text window from the cursor position to the bottom of the window. This routine destroys the contents of A and Y.

---

**CLRSCR**

CLRSCR clears the low-resolution graphics display to black. If you call this routine while the video display is in text mode, it fills the screen with inverse-mode at-sign (@) characters. This routine destroys the contents of A and Y.

---

**CLRTOP**

CLRTOP is the same as CLRSCR, except that it clears only the top 40 rows of the low-resolution display.

---

**COUT**

COUT calls the current character output subroutine (section 3.3.1). The character to be sent to the output device should be in the accumulator. COUT calls the subroutine whose address is stored in CSW (locations \$36 and \$37), which is usually the standard character output COUT1.

---

**COUT1**

COUT1 (section 3.3.2) displays the character in the accumulator on the display screen at the current cursor position and advances the cursor. It places the character using the setting of the inverse mask (location \$32). It handles these control characters: carriage return, line feed, backspace, and bell. When it returns control to the calling program, all registers are intact.

---

**CROUT**

CROUT sends a carriage return to the current output device.

---

**CROUT1**

CROUT1 clears the screen from the current cursor position to the edge of the text window, then calls CROUT.

---

**HLINE**

HLINE draws a horizontal line of blocks of the color set by SETCOL on the low-resolution graphics display. Call HLINE with the vertical coordinate of the line in the accumulator, the leftmost horizontal coordinate in the Y register, and the rightmost horizontal coordinate in location \$2C. HLINE returns with A and Y scrambled and X intact.

---

---

**HOME**

HOME clears the display and puts the cursor in the upper-left corner of the screen.

---

**PLOT**

PLOT puts a single block of the color value set by SETCOL on the low-resolution display screen. Call PLOT with the vertical coordinate of the line in the accumulator, and its horizontal position in the Y register. PLOT returns with the accumulator scrambled, but X and Y intact.

---

**PRBL2**

PRBL2 sends from 1 to 256 blanks to the standard output device. Upon entry, the X register should contain the number of blanks to send. If X = \$00, then PRBLANK will send 256 blanks.

---

**PRBYTE**

PRBYTE sends the contents of the accumulator in hexadecimal to the current output device. The contents of the accumulator are scrambled.

---

**PRERR**

PRERR sends the word ERR, followed by a bell character, to the standard output device. On return, the accumulator is scrambled.

---

**PRHEX**

PRHEX prints the lower nibble of the byte in the accumulator as a single hexadecimal digit. On return, the contents of the accumulator are scrambled.

---

**PRTAX**

PRTAX prints the contents of the A and X registers as a four-digit hexadecimal value. The accumulator contains the first byte printed, and the X register contains the second. On return, the contents of the accumulator are scrambled.

---

---

### ***SCRN***

SCRN returns the color value of a single block on the low-resolution display. Call it with the vertical position of the block in the accumulator and the horizontal position in the Y register. The block's color is returned in the accumulator. No other registers are changed.

---

### ***SETCOL***

SETCOL sets the color used for plotting in low-resolution graphics to the value passed in the accumulator. The colors and their values are listed in Table 5-4.

---

### ***VLINE***

VLINE draws a vertical line of blocks of the color set by SETCOL on the low-resolution display. Call VLINE with the horizontal coordinate of the line in the Y register, the top vertical coordinate in the accumulator, and the bottom vertical coordinate in location \$2D. VLINE returns with the accumulator scrambled.



## 5.9 I/O Firmware Support

Apple IIc video firmware conforms to the I/O firmware protocol (section 3.4.2). However, it does not support windows other than the full 80-by-24 window in 80-column mode, and the full 40-by-24 window in 40-column mode. The video (port 3) protocol table is shown in Table 5-12.

*Table 5-12. Port 3 Firmware Protocol Table*

Address	Value	Description
\$C30B	\$01	Generic signature byte of firmware cards
\$C30C	\$88	80-column card device signature
\$C30D	\$ii	\$C3ii is entry point of initialization routine (PINIT).
\$C30E	\$rr	\$C3rr is entry point of read routine (PREAD).
\$C30F	\$ww	\$C3ww is entry point of write routine (PWRITE).
\$C310	\$ss	\$C3ss is entry point of the status routine (PSTATUS)

---

### ***PINIT***

PINIT does the following:

- Sets a full 80-column window
- Sets 80STORE (\$C001)
- Sets 80COL (\$C00D)
- Switches on ALTCHAR (\$C00F)
- Clears the screen; places cursor in upper-left corner
- Displays the cursor.

---

### ***PREAD***

PREAD reads a character from the keyboard and places it in the accumulator with the high bit cleared. It also puts a zero in the X register to indicate IORESULT = GOOD.

---

### ***PWRITE***

PWRITE should be called after placing a character in the accumulator with its high bit cleared. PWRITE does the following:

- Turns the cursor off.
- If the character in the accumulator is not a control character, turns the high bit on for normal display or off for inverse display, displays it at the current cursor position, and advances the cursor; if at the end of a line, does carriage return but not line feed. See Table 5-13.
- Carries out control functions as shown in Table 5-10.

**Table 5-13. Pascal Video Control Functions**

CONTROL-	Hex	Function performed
E or e	\$05	Turns cursor on (enables cursor display)
F or f	\$06	Turns cursor off (disables cursor display)
G or g	\$07	Sounds bell (beeps)
H or h	\$08	Moves cursor left one column. If cursor was at beginning of line, moves it to end of previous line
J or j	\$0A	Moves cursor down one row; scrolls if needed
K or k	\$0B	Clears to end of screen
L or l	\$0C	Clears screen; moves cursor to upper-left of screen
M or m	\$0D	Moves cursor to column 0
N or n	\$0E	Displays subsequent characters in normal video. (Characters already on display are unaffected.)
O or o	\$0F	Displays subsequent characters in inverse video. (Characters already on display are unaffected.)
V or v	\$16	Scrolls screen up one line; clears bottom line
W or w	\$17	Scrolls screen down one line; clears top line
Y or y	\$19	Moves cursor to upper-left (home) position on screen
Z or z	\$1A	Clears entire line that cursor is on
or \	\$1C	Moves cursor right one column; if at end of line, does CONTROL-M
^ or [	\$1D	Clears to end of the line the cursor is on, including current cursor position; does not move cursor
~ or 6	\$1E	GOTOxy: initiates a GOTOxy sequence; interprets the next two characters as x+32 and y+32, respectively
_	\$1F	If not at top of screen, moves cursor up one line

When PWRITE has completed this, it

- turns the cursor back on (if it was not intentionally turned off).
- puts a zero in the X register (IORESULT = GOOD) and returns to the calling program.

---

### **PSTATUS**

A program that calls PSTATUS must first put a request code in the accumulator: either a 0 (meaning "Ready for output?" or a 1 (meaning "Is there any input?"). PSTATUS returns with the reply in the carry bit: 0 (*No*) or 1 (*Yes*). If the request was not 0 or 1, PSTATUS returns with a 3 in the X register (IORESULT = ILLEGAL OPERATION); otherwise, PSTATUS returns with a 0 in the X register (IORESULT = GOOD).

# *Disk Input and Output*



The external disk drive connector is described in section 11.10.

The Apple IIc supports both its built-in disk drive and an optional external drive; both drives use single-sided, 35-track, 16-sector format. The disk I/O port characteristics are summarized in Table 6-1.

The firmware resides in the \$C600 address space. It supports the built-in drive as if it were slot 6 drive 1, and the external drive as if it were slot 6 drive 2. If disk startup is unsuccessful, the firmware shuts off the disk drive motor and displays the message, Check Disk Drive on the display screen.

Table 6-1. Disk I/O Characteristics

Port Number	I/O Port 6 Drive 1 (built-in drive) I/O Port 6 Drive 2 (external drive)
Commands	IN#6 or PR#6 CALL -151 (to get to the Monitor from BASIC), then (6) (CONTROL)-(K) or (6) (CONTROL)-(P)
Initial Characteristics	All resets except (CONTROL)-(RESET) with a valid reset vector eventually pass control to the built-in disk drive.
Hardware Location	Description
\$C0E0-EF	Reserved
Monitor Firmware Routines	None
I/O Firmware Entry Points	\$C600 (port 6)
Use of Screen Holes	Port 6 main and auxiliary memory screen holes are reserved.

## 6.1 Startup

A power-on startup, an **(G)-(CONTROL)-(RESET)** startup, or a **(CONTROL)-(RESET)** startup that does not find a valid reset vector results in a cold start. The cold-start routine first sets a number of soft switches (see Chapter 2) and then passes control to the program entry point at \$C600. This code turns on the internal drive motor, recalibrates the read/write head at track zero, then reads sector zero from that track. The sector contents are loaded and decoded starting at address \$800; then program control passes to \$801. This loaded program varies depending on the operating system or application program on the disk.

To restart the system, issue a **PR#6** command from BASIC, **(6) (CONTROL)-(P)** from Monitor command mode, or **JMP \$C600** from a machine-language program.

## 6.2 External Drive Startup

The ProDOS operating system (but not the DOS or Pascal operating systems) supports startup using the external disk drive. This ProDOS feature makes it possible to start the Apple IIc with a diagnostic program in the event that the built-in drive does not work.

To restart using the external drive, insert a ProDOS disk in the external drive, then invoke the Monitor (**CALL -151**) and issue **(7) (CONTROL)-(P)**.

**Remember:** External drive startup works with ProDOS-based programs, but not with Pascal 1.1 or 1.0, or with DOS.



# *Serial I/O Port 1*

If you need to change port characteristics from a program, read section 7.6 for the methods to use.

Serial port 1 is the first of two serial I/O ports available on the Apple IIc. It is intended primarily as an output port for RS-232 devices, such as printers and plotters. It can be changed to a serial communication port (like port 2) using the System Utilities Disk.



---

**Warning**

*Although the Apple IIc serial ports are similar to the Apple IIe Super Serial Card, there are many important differences. Refer to Appendix F for a summary of these differences.*

---

If you change port 1 to a communication port, refer to the descriptions in Chapter 8, and use 1 instead of 2 for the port number when required.

Table 7-1 summarizes the characteristics of this port if used as a printer/plotter port, and is a guide to the other information in this chapter.

The serial port back panel connectors are described in section 11.11.



*Table 7-1. Serial Port 1 Characteristics*

Port Number	Serial port 1
Commands	Keyboard command PR#1
	BASIC command PR#1
	Monitor command 1 CONTROL-P (does not work if there is an operating system in RAM)
	All other commands Table 7-2
Initial Characteristics	Table 7-3
Hardware Page Locations	Table 7-4
Monitor Firmware Routines	None
I/O Firmware Entry Points	Table 7-5
Use of Screen Holes	Table 7-6
Use of Other Pages	None

## 7.1 Using Serial Port 1

You can access the firmware from BASIC in the usual way—that is, by issuing CONTROL-D (if DOS or ProDOS is in RAM) and PR#1. Subsequent output is directed to the printer (or other device) connected to serial port 1.

To direct Pascal output to the printer, you can use either #6: or PRINTER: .

Table 7-2 lists the commands you can use with serial port 1, either from a program or from the keyboard, after you issue PR#1. Each command must be preceded by CONTROL-I (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command.

You do not have to press **RETURN** after commands.

**Note:** The commands themselves are letter commands, not control characters.

Refer to Table 7-5 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

**Table 7-2. Printer Port Commands**

Command	Description		
nnn	Set new line width of nnn (from 1 through 255). This command must be followed by N (see below) or by a carriage return.		
nnB	Set baud rate to value corresponding to nn:		
nn	Rate	nn	Rate
1	50	9	1800
2	75	10	2400
3	110 (109.92)	11	3600
4	135 (134.58)	12	4800
5	150	13	7200
6	300	14	9600
7	600	15	19200
8	1200		
nD	Set data format to values corresponding to n:		
n	Data Bits	Stop Bits	
0	8	1	
1	7	1	
2	6	1	
3	5	1	
4	8	2	
5	7	2	
6	6	2	
7	5	2	
I	Echo printer output on the screen.		
K	Disable automatic line feed after carriage return.		
L	Generate line feed after carriage return.		
nnnN	Change line width to nnn (from 1 through 255; nnn is optional); do not echo printer output on the screen. <b>Note:</b> 0N does not disable automatic generation of carriage return; to do so, use Z command, put 0 directly in location \$579 or use System Utilities Disk.		

**Table 7-2—Continued. Printer Port Commands**

Command	Description
nP	Set parity corresponding to n:
n	Parity
0	None
1	Odd
2	None
3	Even
4	None
5	MARK (1)
6	None
7	SPACE (0)
R	Reset port 1 (Table 7-3) and exit from serial port 1 firmware.
S	Send a 233 millisecond BREAK character (used with some printers to synchronize with serial ports).
Z	Zap (ignore) further command characters (until <b>CONTROL-RESET</b> or PR#1). Do not format output or insert carriage returns into output stream.

The command character starts off as CONTROL-I for the printer port. You can change it to a different control character by typing the current control character followed immediately by the new control character you want. This is useful if you want to be able to send CONTROL-I to the printer without firmware intervention. For example, to change the command character from CONTROL-I to CONTROL-V, simply press **CONTROL-I** **CONTROL-V**. (CONTROL-V and CONTROL-W are the recommended substitute control characters.) To change the command character back again, press **CONTROL-V** **CONTROL-I**.

Do not use **CONTROL-A**, **CONTROL-B**, **CONTROL-C**, **CONTROL-H**, **CONTROL-J**, **CONTROL-L**, **CONTROL-M** or **CONTROL-Y**. Apple IIc firmware may intercept these control characters, causing unpredictable results.

The following are examples of valid commands and command sequences.

Echo output to the display screen:

**CONTROL-I I**

Set line width 72, disable line feed, and echo:

**CONTROL-I K CONTROL-I 72N**

Change control character to CONTROL-V:

**CONTROL-I CONTROL-V RETURN**

An example of how you can send CONTROL-I as part of a character stream:

**CONTROL-V (command) RETURN**

---

## 7.2 Characteristics at Startup

After power-up, the printer firmware sets the configuration given in Table 7-3. These values are stored in the auxiliary-memory screen holes (Table 7-6).

*Table 7-3. Initial Characteristics of Printer Port*

9600 baud

Eight data bits, no parity bits, two stop bits

80-column line width; no echo to display screen

Firmware supplies line feed after carriage return.

Command character is set to CONTROL-I (see below).

You can change some of these settings from the keyboard by typing PR#1, the command character, and one of the commands listed in Table 7-2. Section 7.6 describes how port characteristics change as a result of various activities.

**Note:** You can type more than one command on a line, but each must be preceded by the command character.

## 7.3 Hardware Page Locations

ACIA stands for **Asynchronous Communication Interface Adapter**, a serial I/O chip. Note in Chapter 11 that some of the bit assignments for this port differ from those for port 2.

Table 7-4 lists for serial port 1 the addresses and bit assignments of its hardware registers on page \$C0. The registers are internal to a 6551 ACIA; their bit assignments are described in section 11.11.

*Table 7-4. Serial Port 1 Hardware Page Locations*

Location	Description
\$C090-\$C097	Reserved
\$C098	ACIA transmit/receive data register
\$C099	ACIA status register
\$C09A	ACIA command register
\$C09B	ACIA control register
\$C09C-\$C09F	Reserved

## 7.4 I/O Firmware Support

Table 7-5 lists the locations and values of the I/O firmware protocol table. This standardized protocol is available for use by any application program. Section 3.4.2 describes how to use this protocol.

*Table 7-5. Port 1 I/O Firmware Protocol*

Address	Value	Description
\$C105	\$38	Pascal ID byte
\$C107	\$18	Pascal ID byte
\$C108	\$01	Generic signature byte of firmware cards
\$C10C	\$31	Same ID as for Super Serial Card
\$C10D	\$11	\$C11i is entry point of initialization routine (PINIT).
\$C10E	\$1r	\$C11r is entry point of read routine (PREAD).
\$C10F	\$1w	\$C11w is entry point of write routine (PWRITE).
\$C110	\$1s	\$C11ss is entry point of the status routine (PSTATUS).
\$C111	non-0	No optional routines



## 7.5 Screen Hole Locations

The ACIA register bits are defined in Chapter 11.

Table 7-6 lists the screen hole locations that serial port 1 uses. Note that the auxiliary-memory locations are reserved for startup value settings, which are listed and interpreted in the table.

**Table 7-6. Serial Port 1 Screen Hole Locations**

**Auxiliary Memory Screen Holes** (firmware loads at power-up reset):


Location	Description										
\$478	\$9E (ACIA control reg: 8 data + 2 stop bits, 9600 baud)										
\$479	\$0B (ACIA command reg: no parity)										
\$47A	\$40 (flags: no echo, auto LF after CR, serial port)										
	<table><tr><th>Bit</th><th>Interpretation</th></tr><tr><td>7</td><td>Echo output on display (0 = no echo)</td></tr><tr><td>6</td><td>Generate LF after CR (0 = no LF)</td></tr><tr><td>5-1</td><td>Always = 0 (reserved)</td></tr><tr><td>0</td><td>1 = communication port; 0 = serial printer port</td></tr></table>	Bit	Interpretation	7	Echo output on display (0 = no echo)	6	Generate LF after CR (0 = no LF)	5-1	Always = 0 (reserved)	0	1 = communication port; 0 = serial printer port
Bit	Interpretation										
7	Echo output on display (0 = no echo)										
6	Generate LF after CR (0 = no LF)										
5-1	Always = 0 (reserved)										
0	1 = communication port; 0 = serial printer port										
\$47B	\$50 (printer width: 80 columns)										
	<table><tr><th>Bit</th><th>Interpretation</th></tr><tr><td>7-0</td><td>Printer width (0 = do not insert CR)</td></tr></table>	Bit	Interpretation	7-0	Printer width (0 = do not insert CR)						
Bit	Interpretation										
7-0	Printer width (0 = do not insert CR)										

**Main Memory Screen Holes:**

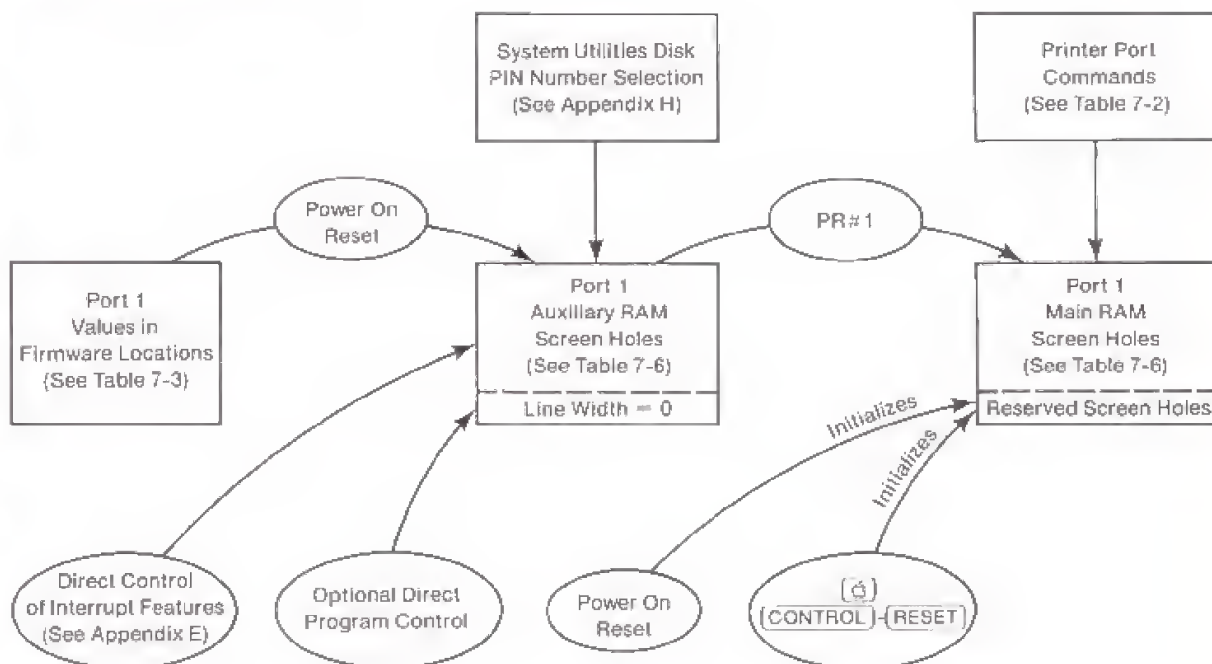
Location	Description
\$479	Reserved
\$4F9	Reserved
\$579	Printer width (1 - 255; 0 = disable formatting)
\$5F9	Temporary storage location
\$679	Bit 7 = 1 while the firmware is parsing a command string.
\$6F9	Current command character (initially CONTROL-I)
\$779	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return.
\$7F9	Current printer column

## 7.6 Changing Port Characteristics

Figure 7-1 is a diagram of where the port characteristics are stored and moved under different circumstances. As you can see from the figure:

- When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed in Table 7-3 from ROM into the auxiliary memory screen holes listed in Table 7-6.
- If you specify new characteristics using the System Utilities Disk, the SUD software changes the values in the auxiliary memory screen holes.
- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either -CONTROL-RESET or a simple CONTROL-RESET. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.
- PR#1 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.
- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location \$579.
- The firmware uses the port as it is defined in the main memory screen holes at any given time. You should use the commands listed in Table 7-2 to change them.

**Figure 7-1. Port Characteristics**



### 7.6.1 Data Format and Baud Rate

Serial data transfer consists of a string of ones and zeros sent down a wire at a prearranged rate of speed, called the **baud rate**. With most equipment, baud simply equals the number of bits per second.

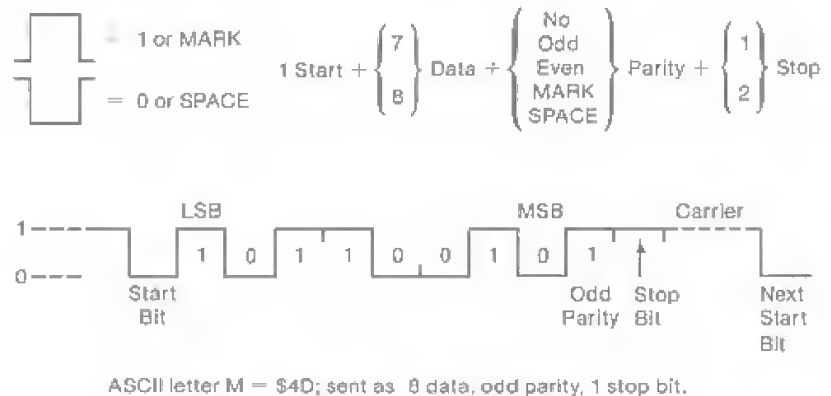
Before transfer begins, both sender and receiver look for a continuous value of 1: this is called the **carrier** (Figure 7-2). When the value goes to zero, the receiver presumes it is a **start bit**—that is, the bit that designates the beginning of a character of data. If it lasts longer than a bit could possibly last, it is considered a **BREAK** signal, which some printers use for synchronization.

If the first zero proves to be a bit, it is interpreted as the start bit. Next come the 7 or 8 data bits (6 is seldom used with computers), low-order bit first. If parity is on, it comes next in the message. Finally, one or two **stop bits** (with a value of 1) appear. The stop bits have a value of 1, like the carrier. The next start bit begins transfer of the next character of data.

The **parity bit** provides a simple check of data validity. Odd parity means the sender counts the number of ones among the data bits, and sends the appropriate parity bit to make the total number of ones odd. With even parity, the sender adds the appropriate parity bit to make the total number of one-bits even. MARK parity is always a 1 bit; SPACE parity is always a zero. The receiver can then check that the parity bit is correct.

If the baud rate is 300, and the data format is 1 start bit plus 7 data bits plus 1 parity bit plus 1 stop bit, then the actual transfer rate is about 30 characters per second.

**Figure 7-2. Data Format**



### 7.6.2 Carriage Return and Line Feed

If you are using a typewriter and you push the carriage all the way to the right (in other words, position the printing mechanism at the left margin), you have performed a **carriage return**. On the other hand, turning the platen so the paper moves to the next line (or using the index key on an electric typewriter) is called a **line feed**. Most typewriters perform a line feed automatically after a carriage return, and so the two seem to be one—but they are not.

Carriage return and line feed are separate ASCII codes. Carriage return is sometimes denoted CR; it is ASCII code 13 (\$0D). Line feed, sometimes denoted LF, is ASCII code 10 (\$0A).  $\text{⏏}$  on the Apple IIc keyboard generates a LF.

Some printers can supply a line feed automatically after detecting a carriage return; others cannot. If the printer does not supply a line feed after a carriage return and it is not supplied in the data stream, the printer will keep printing over on the same line. On the other hand, if both the printer and the Apple IIc firmware supply LF after CR, double line-spacing will result.

If the print head keeps moving too far to the right across the page and then prints many characters on top of one another on the right, then the firmware should be instructed to furnish CR after a certain line width has been reached. If the printer prints too short a line before moving to the next line, then probably the firmware is using too small a line width.

If the printer misses characters at the beginning of each line but otherwise prints correctly, there is probably not enough time for the print mechanism to return to the left margin in response to CR. You must use a lower baud rate with such a printer.

---

### ***7.6.3 Sending Special Characters***

If you want to send special characters (control characters) to the printer without having them intercepted and executed by the Apple IIc firmware, use the Z command. If the only special character that causes a problem is the command character (normally CONTROL-I for port 1), you can change just the command character instead of using the zap (Z) command. If you use the zap command, the firmware does no formatting; that is, it does not check line width or insert carriage returns or line feeds. This may be necessary to send graphics to a printer or plotter.

---

### ***7.6.4 Displaying Output on the Screen***

You can display printer output on the screen, but if the printer line width exceeds the 40 or 80 columns you have selected for display, you should turn off video display.



# *Serial I/O Port 2*

Serial port 2 is the second of two serial I/O ports available on the Apple IIc. It is intended primarily as a communication port for modems. You can change it to a serial printer port (like port 1) using the System Utilities Disk. If you need to change port characteristics from a program, read section 8.6.



---

**Warning**

*Although the Apple IIc serial ports are similar to the Apple Super Serial Card, there are many important differences. Refer to Appendix F for a summary of these differences.*

---

If you change port 2 to a serial printer port, refer to the descriptions in Chapter 7, and use 2 instead of 1 for the port number when required.

Table 8-1 summarizes the characteristics of this port and is a guide to the other information in this chapter.

The serial port connectors are described in section 11.11.

**Table 8-1. Serial Port 2 Characteristics**

<b>Port Number</b>	Serial Port 2
<b>Commands</b>	<p>Keyboard commands</p> <p>IN#2 before Table 8-2 commands</p> <p>IN#2 to accept port 2 input</p> <p>PR#1 to echo input to printer</p> <p>PR#2 to echo input back to port 2</p> <p>BASIC commands (same)</p> <p>Monitor command</p> <p>(2) (CONTROL)-P (This command works only if there is no operating system in RAM.)</p> <p>All other commands</p> <p>Table 8-2</p>
<b>Initial Characteristics</b>	Table 8-3
<b>Hardware Page Locations</b>	Table 8-4
<b>Monitor Firmware Routines</b>	None
<b>I/O Firmware Entry Points</b>	Table 8-5
<b>Use of Screen Holes</b>	Table 8-6
<b>Use of Other Pages</b>	In terminal mode, firmware uses auxiliary memory locations \$800-\$87F to store keyboard input, and \$880-\$8FF as a serial input buffer.

## 8.1 Using Serial Port 2

You can access the firmware from BASIC in the usual way—that is, by issuing CONTROL-D (if DOS or ProDOS is in RAM) followed by IN#2 or PR#2. Subsequent input and output are routed through the modem (or other device) connected to serial port 2.

**Note:** In terminal mode, the modem port commands listed in Table 8-2 must follow CONTROL-D and IN#2 (*not* PR#2) and the command character (which is usually CONTROL-A).

Refer to Table 8-5 for the standard firmware entry points that Pascal 1.1 and 1.2 use.

To transfer files to the modem under Pascal, specify REMOUT: or #8: . To transfer files from the modem under Pascal, specify REMIN: or #7: .

Table 8-2 lists the commands you can use with serial port 2, either from a program or from the keyboard, after you issue IN#2. Each command must be preceded by CONTROL-A (the command character). As soon as you issue the command character, the serial port firmware displays a flashing question mark cursor to indicate it is awaiting a command. If you press **(RETURN)**, you get the current video cursor again.

You do not have to press **(RETURN)** after commands.

**Note:** The commands themselves are letter commands, not control characters.

**Table 8-2. Modem Port Commands**

Command	Description		
nnn	Set new line width of nnn (from 1 through 255); this must be followed immediately by N (see below) or by carriage return.		
nnB	Set baud rate to value corresponding to nn:		
nn	Rate	nn	Rate
1	50	9	1800
2	75	10	2400
3	110 (109.92)	11	3600
4	135 (134.58)	12	4800
5	150	13	7200
6	300	14	9600
7	600	15	19200
8	1200		
nD	Set data format to values corresponding to n:		
n	Data Bits	Stop Bits	
0	8	1	
1	7	1	
2	6	1	
3	5	1	
4	8	2	
5	7	2	
6	6	2	
7	5	2	



*Table 8-2—Continued. Modem Port Commands*

Command	Description
I	Echo output on the screen.
K	Disable automatic line feed after carriage return.
L	Generate line feed after carriage return.
nnnN	Set line width to nnn (from 1 through 255); do not echo output on the screen. <b>Note:</b> 0N does not turn off automatic generation of carriage return; to do so, use Z command, put 0 directly in location \$57A, or use the System Utilities Disk.
nP	Set parity corresponding to n:
n	Parity
0	None
1	Odd
2	None
3	Even
4	None
5	MARK (1)
6	None
7	SPACE (0)
Q	Quit terminal mode.
R	Reset port 2 (Table 8-3) and exit from serial port 2 firmware.
S	Send a 233 millisecond BREAK character.
T	Enter terminal mode. Use this command after IN#2 only. Also, if you follow this command by PR#2, the Apple IIc will echo input to output. (If the other device does so too, the first character entered will loop endlessly, locking up the system. Use <b>CONTROL-RESET</b> to get out.)
Z	Zap (ignore) further command characters until <b>CONTROL-RESET</b> . Do not format output or insert carriage returns into output stream.
<b>CONTROL-T</b>	This command from a remote device puts the Apple IIc in terminal mode if IN#2 is already in effect. It is the same as <b>CONTROL-A T</b> typed locally.
<b>CONTROL-R</b>	This command from a remote device undoes the terminal mode command. If IN#2 and PR#2 are in effect, the remote keyboard and display become the input and output devices of the local Apple IIc. It is the same as <b>CONTROL-A Q</b> typed locally.

The command character starts off as CONTROL-A for the communication port. You can change it to a different control character by typing the current control character followed immediately by the new control character you want. This is useful if you want to be able to send CONTROL-A to the output device without firmware intervention.

For example, to change the command character from CONTROL-A to CONTROL-V, simply press (CONTROL)-(A) (CONTROL)-(V). (CONTROL-V and CONTROL-W are the recommended substitute control characters.) To change the command character back again, press (CONTROL)-(V) (CONTROL)-(A).



---

**Warning**

*Do not use (CONTROL)-(B), (CONTROL)-(C), (CONTROL)-(H), (CONTROL)-(I), (CONTROL)-(J), (CONTROL)-(L), (CONTROL)-(M) or (CONTROL)-(Y): Apple IIc firmware may intercept these control characters, causing unpredictable results.*

---

The following are examples of valid commands and command sequences.

Enable echo to the screen:

(CONTROL)-(A) (I)

Send a BREAK character to a remote device:

(CONTROL)-(A) (B)

Change the control character to CONTROL-V (For example, so you can send CONTROL-A as part of a character stream.):

(CONTROL)-(A) (CONTROL)-(V) (CONTROL)-(V) (command)

## 8.2 Characteristics at Startup

After power-up, the firmware sets the configuration given in Table 8-3. These values are stored in the auxiliary-memory screen holes (Table 8-6).

*Table 8-3. Initial Characteristics of Communication Port*

300 baud

Eight data bits, no parity bits, one stop bit

Firmware does not supply line feed after carriage return.

Firmware does not insert carriage returns into output stream.

Firmware does not echo output to the display screen.

Command character is set to CONTROL-A.

You can change some of these settings from the keyboard using the command character followed by one of the commands listed in Table 8-2. Section 8.6 describes how port characteristics change as a result of various activities.

If you change any of these values using keyboard commands or commands from a program, subsequent accesses to the port firmware (even by another program) use the new settings instead of the power-up values. This allows you to change the settings once at system startup, and get the desired configuration for subsequent uses. Refer to section 8.6 for a complete description of these processes.

## 8.3 Hardware Page Locations

ACIA stands for Asynchronous Communication Interface Adapter, a serial I/O chip. Note in Chapter 11 that some of the bit assignments for this port differ from those for port 1.

Table 8-4 lists for serial Port 2 the addresses of its hardware registers on page \$C0. The registers are internal to a 6551 ACIA; their bit assignments are described in section 11.11.

**Table 8-4. Serial Port 2 Hardware Page Locations**

Location	Description
\$C0A0-\$C0A7	Reserved
\$C0A8	ACIA transmit/receive data register
\$C0A9	ACIA status register
\$C0AA	ACIA command register
\$C0AB	ACIA control register
\$C0AC-\$C0AF	Reserved

## 8.4 I/O Firmware Support

Table 8-5 lists the values in the I/O firmware protocol table for serial port 2. This standardized protocol is available for use by any application program. Section 3.4.2 describes how to use this protocol.

**Table 8-5. Port 2 I/O Firmware Protocol**

Address	Value	Description
\$C205	\$38	Pascal ID byte
\$C207	\$18	Pascal ID byte
\$C20B	\$01	Generic signature byte of firmware cards
\$C20C	\$31	Same ID as for Super Serial Card
\$C20D	\$ii	\$C1ii is entry point of initialization routine (PINIT).
\$C20E	\$rr	\$C1rr is entry point of read routine (PREAD).
\$C20F	\$ww	\$C1ww is entry point of write routine (PWRITE).
\$C210	\$ss	\$C1ss is entry point of the status routine (PSTATUS).
\$C211	non-0	No optional routines

## 8.5 Screen Hole Locations

The ACIA register bits are defined in Chapter 11.

Table 8-6 lists the screen hole locations that serial port 2 uses. Note that the auxiliary-memory locations are reserved for startup value settings, which are listed and interpreted in the table.

*Table 8-6. Serial Port 2 Screen Hole Locations*

**Location      Description**

**Auxiliary Memory Screen Holes** (firmware loads values at power-up):

\$47C	\$16 (ACIA control reg: 8 data + 1 stop bit, 300 baud)
\$47D	\$0B (ACIA command reg: no parity)
\$47E	\$01 (flags: no echo, no auto LF after CR, communication port)

Bit	Interpretation
7	Echo output on display (0 = no echo)
6	Generate LF after CR (0 = no LF)
5-1	Always = 0 (reserved)
0	1 = communication port; 0 = serial printer port

\$47F	\$00 (line length: do not add any CR to output stream)
-------	--

Bit	Interpretation
7-0	Line length (0 = do not insert CR)

**Main Memory Screen Holes:**

\$47A	Reserved
\$4FA	Reserved
\$57A	Line length (1 - 255; 0 = disable formatting)
\$5FA	Temporary storage location
\$67A	Bit 7 = 1 if and only if the firmware is currently parsing a command string.
\$6FA	Current command character (initially CONTROL-I)
\$77A	Bit 7 = 1 if echo to display is on; bit 6 = 1 if firmware is to generate a line feed after carriage return.
\$7FA	Current column

## 8.6 Changing Port Characteristics

Figure 8-1 is a diagram of where the port characteristics are stored and moved under different circumstances. As you can see from the figure:





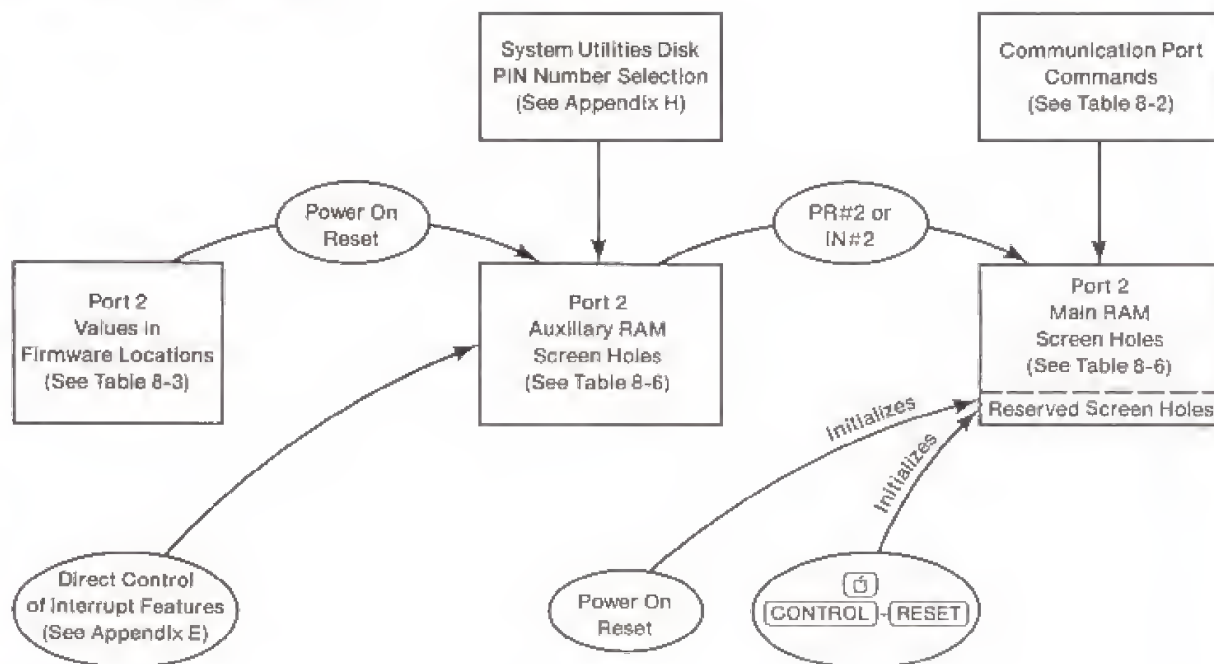
- When the power is first turned on, the Monitor reset firmware moves the predefined set of port characteristics listed in Table 8-2 from ROM into the auxiliary memory screen holes listed in Table 8-6.
- If you specify new characteristics using the System Utilities Disk, the utility software changes the values in the auxiliary memory screen holes.
- The values stored in the auxiliary memory screen holes are affected by power-on reset, but not by either -CONTROL- or a simple -CONTROL-. This feature is provided so that a port that has been reconfigured will remain that way while some other program (such as an application program) is started up.
- IN#2 causes the firmware to move the characteristics stored in the auxiliary memory screen holes into the main memory screen holes.
- A program can change values in the main memory screen holes directly. However, the only value guaranteed to be in the same place for the entire Apple II series is the line length in main memory location \$57A.
- The firmware uses the port as it is defined in the main memory screen holes at any given time. You should use the commands listed in Table 8-2 to change these characteristics.



Figure 8-1. Port 2 Characteristics



### 8.6.1 Data Format and Baud Rate

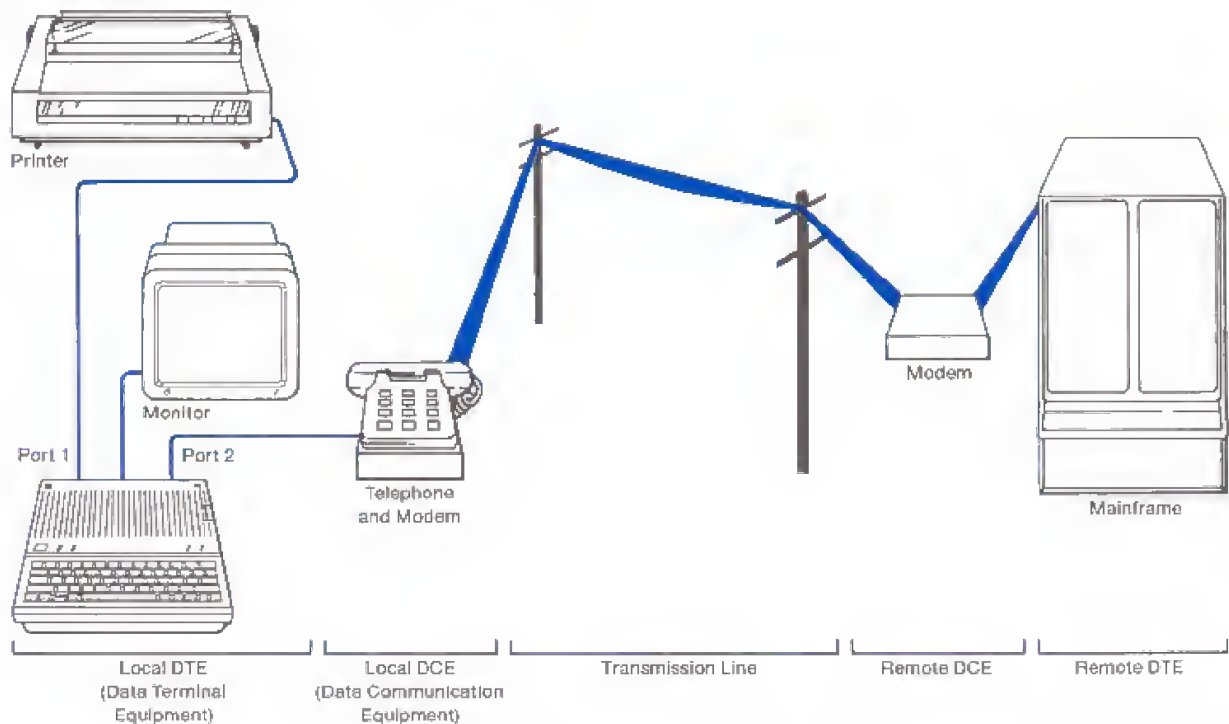
Section 7.6.1 describes data format and baud rate, and explains how they apply to printers. Refer to that section or to the Glossary for the definition of terms.

A noteworthy characteristic of data communication is its strangeness: sometimes the oddest changes make a given communication arrangement work or not work. You must keep this notion firmly in mind when working with serial port 2. For example, modem communication involves quite a few elements (Figure 8-2):

- the Apple IIc and its firmware, with the baud rate, data format, and other characteristics you have selected
- the cable from the Apple IIc to the modem
- the modem
- possibly an acoustic coupler for a telephone handset

- the telephone lines, with their switching equipment, boosters, and noise
- some combination of modem, cable, and computer or terminal on the other end.

**Figure 8-2.** *Devices in a Typical Communication Setup*



As you can imagine, some method is required for success. If you have problems, change only one variable at a time, and then cycle through the other variables one at a time. Take nothing for granted. The data format advertised for an information service, for example, may be different from the one you end up using with the Apple IIc.

### **8.6.2 Carriage Return and Line Feed**

If you are communicating with a computer or terminal, carriage return and line feed may or may not be involved. Start off without generating them, and turn on automatic generation only as needed. They are described as used with printers in section 7.6.2.

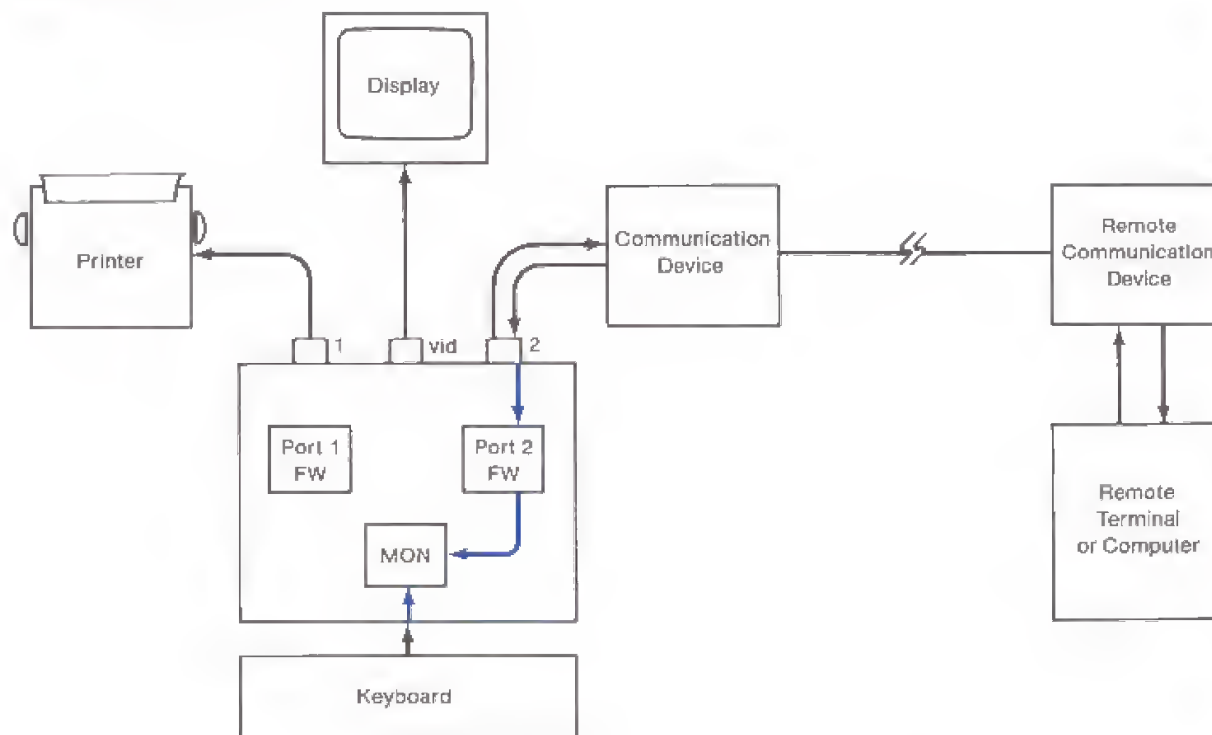
---

### ***8.6.3 Routing Input and Output***

This section discusses the possible ways that serial port 2 can route information. Sometimes the cause of communication problems is that information is not going where you think it is, or it is and you cannot see evidence of the fact. Figures 8-3 through 8-6 show some of the patterns of information flow you can select. This section and the following subsections tell you how to use them.

It is best to read all of this material as a unit: questions that arise while you read one description may be answered elsewhere.

Figure 8-3. Effect of IN#2



The simplest serial port 2 command is IN#2 (Figure 8-3). Port 2 becomes the input device. Data coming into the port gets passed to the input buffer (page 2 of main memory). Applesoft firmware and system software can see the data and carry out commands in the normal way.

Of course, you can also use just the PR#2 command—for example, if you want to send a listing to the modem.

To use port 2 for data communication, you ordinarily put it into terminal mode. Following IN#2, typing **CONTROL-A** gets the attention of the port 2 firmware, which displays a blinking question mark (?) as a prompt. Now type **T** to put the computer in terminal mode. In this mode, the firmware displays a blinking underscore character (\_) as a prompt.

In the discussion that follows, **local** refers to your Apple IIc. **Remote** refers to some other device, usually in a distant location and at the other end of a communication link. The remote device can be any ASCII-generating unit: a terminal or a computer.

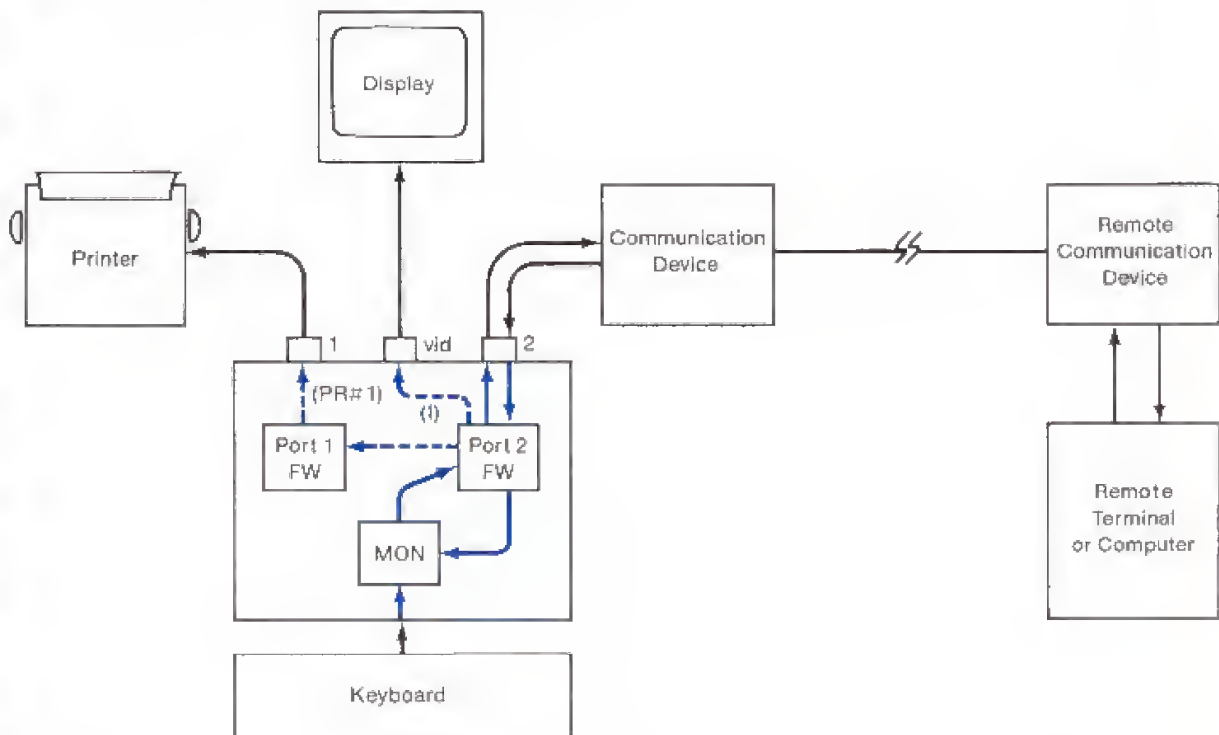
For a further description of what terminal mode does and how to get into and out of it, refer to the last section of this chapter.

If a remote computer is another Apple IIc or an Apple II series machine with a Super Serial Card in it, then most of the commands described here apply to it as well.

### Half Duplex Operation

In half-duplex operation, information can flow from A to B or from B to A, but in only one direction at a time. In a half-duplex setup, the host does not echo back to the terminal what the terminal sends it. For half-duplex operation, use IN#2 and **(CONTROL)-(A) (T)** (Figure 8-4) whether the Apple IIc is the host or the terminal.

Figure 8-4. Effect of IN#2 and T Command (Half Duplex)



IN#2 plus **(CONTROL)-(A) (T)** is the best way to set up the computer for auto-answer operation. The T command allows port 2 firmware to exchange information with the local modem without interference from the local firmware or system software. (The remote device can always cancel the T command with **(CONTROL)-(R)** if necessary, and restore

terminal mode with (CONTROL)-(T).) Avoiding PR#2 at this point means that the Apple IIc can operate as a half-duplex terminal, half-duplex host, or full-duplex terminal. (The remote device can also issue (CONTROL)-(A) PR#2 if PR#2 is required at the local computer.)

In half-duplex operation, the output hook is available for other uses. For example, you can issue PR#1 to print incoming messages from port 2. Use the (CONTROL)-(A) (I) command to display information on the screen.

---

### ***Full Duplex Operation***

In full-duplex operation, information can flow from A to B and from B to A simultaneously. Typically, one of the computers (the host computer) echoes its input to output, so the other computer (the terminal) can easily verify that the communication is taking place.

Figure 8-5 shows the flow of information when the Apple IIc is a full-duplex terminal. (The setup commands, IN#2 and (CONTROL)-(A) (T), are the same as for half duplex.)

If your Apple IIc is the terminal in full-duplex operation, use the N command to turn off echoing input to the screen. If the Apple IIc does echo input to the screen in this setup, everything you type will appear twice: once from the Apple IIc and once from the host computer.

In this mode of operation, if you echo input to the printer you can get a printed record of both sides of the communication session: the input from the host, and the Apple IIc output as echoed by the host.



Figure 8-5. Effect of IN#2 and T Command (Full Duplex Terminal)

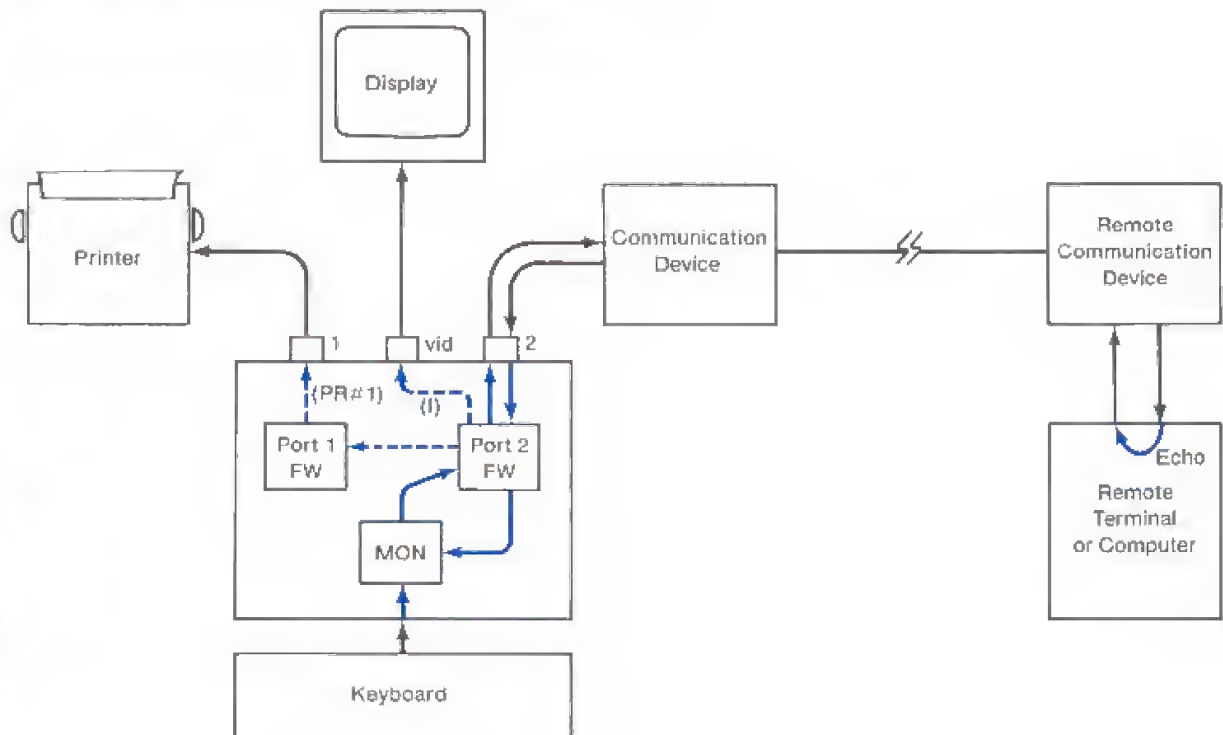


Figure 8-6 shows the flow of information when the Apple IIc is a full-duplex host. In this case, the local Apple IIc must echo input to output for the remote device. The setup commands include PR#2 in this case.



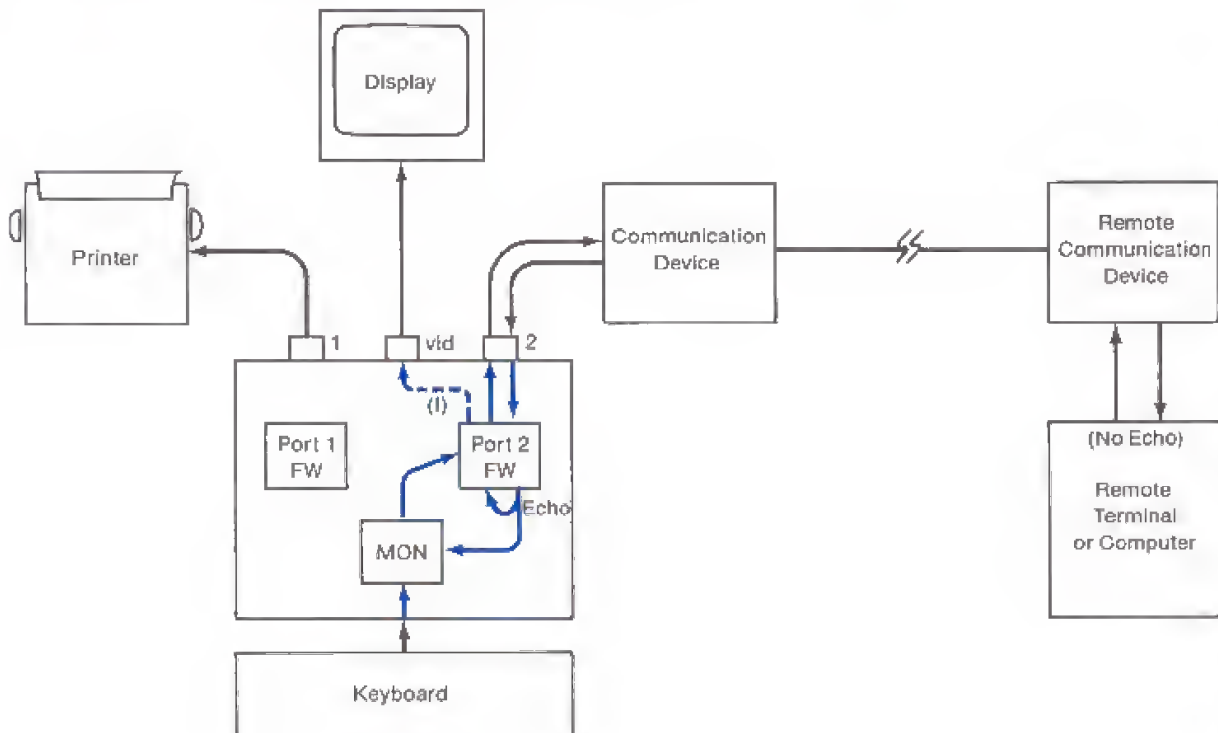
### Warning

*If the Apple IIc echoes input to output and the other computer does too, then the first subsequent keypress will echo back and forth endlessly and lock up the Apple IIc. This will require a **CONTROL**-**RESET** to get out.*

*If you echo input to output when using an information service, the host will end up seeing the echo of what it sent you as though you had typed it.*

In this arrangement, the local output hook is not available for using the printer or other device. To display keyboard and port 2 input on the screen, issue **CONTROL**-**A** **I**.

Figure 8-6. Effect of IN#2, PR#2 and T Command (Full Duplex Host)



---

### Terminal Mode

Terminal mode makes the Apple IIc act like a **dumb terminal**—one that just sends and receives information, but does not process it. Input and output flow through special serial I/O buffers on page 8 of auxiliary memory. Applesoft firmware and system software cannot see or interpret the data; only the serial port 2 firmware deals with it.

In most terminal mode setups, the firmware will not display port 2 input unless you use the **(CONTROL)-(A) (I)** command.

---

### Warning

*When using terminal mode, \$800-\$8FF of auxiliary RAM is used for buffering. Any data stored there will be overwritten when terminal mode is enabled.*

---

**(CONTROL)-(A) (T)** turns on terminal mode, and **(CONTROL)-(A) (Q)** turns it off.

The remote device can go into terminal mode, and then turn off the local Apple IIc's terminal mode with the **(CONTROL)-(R)** command. If it then issues **(CONTROL)-(A) PR#2**, local output will go to the remote device. The remote keyboard and display then become the input and output devices of the local Apple IIc processor. This is remote mode.

In remote mode, the local Apple IIc does not use the serial I/O buffers (as it does in terminal mode); therefore, local firmware and system software detect and interpret all input and output data. So, for example, if you type **CATALOG** at the remote device keyboard, the local Apple IIc will execute the command and list the disk catalog on the remote device's display. (In terminal mode, the local computer would simply display the word **CATALOG** on its screen.)

The remote device can turn the local Apple IIc's terminal mode back on with **(CONTROL)-(T)**. **(CONTROL)-(A) (T)** typed at the remote device only turns on the remote device's terminal mode, unless the command character there has already been changed to something else.

# *Mouse and Game Input*

Section 9.1.6 gives an example of how to use the mouse as a hand control.

This chapter describes the mouse port and hand control (game) input capabilities of the Apple IIc. The mouse and hand controls use the same 9-pin connector on the back panel; the firmware uses the port as directed by keyboard or program commands.

A program can tell if a hand control is connected (section 9.2) but not if a mouse is connected, unless the computer user moves it.

## 9.1 Mouse Input

Table 9-1 is a summary of the characteristics of the mouse port and a guide to the other information in this part of the chapter.

### Warning

*If you want to ensure compatibility with mouse operation on the Apple IIe and other Apple II series computers, always use the I/O firmware entry points listed in Tables 9-4 and 9-5, rather than dealing with mouse hardware and RAM locations directly.*

The mouse back panel connector is described in section 11.12.

*Table 9-1. Mouse Input Port Characteristics*

Port Number	Mouse Input Port 4
BASIC commands	Turn on mouse: <code>PRINT CHR\$(4)"PR#4";PRINT CHR\$(1)</code> Turn off mouse interrupts: <code>PRINT"PR#4";PRINT CHR\$(0)</code> Turn on graphics character set: see section 5.2.2.
Initial Characteristics	After a reset, all mouse interrupts are off, and the rising edge of X0 and Y0 are selected for interrupts.
Hardware Page Locations	Table 9-2
Monitor Firmware Routines	None
I/O Firmware Entry Points	Table 9-3 and Table 9-4
Use of Screen Holes	Table 9-5

### 9.1.1 Mouse Connector Signals

The mouse uses the same 9-pin D-type miniature connector as the hand controls. However, the interpretation of the signals arriving on the pins differs depending on the commands and signals received. The names of the pin assignments when a mouse is connected are shown in Figure 11-37.



---

### **9.1.2 Mouse Operating Modes**

Later sections of this chapter describe how to set various modes for mouse operation. This section tells what the modes are for.

In all the interrupt modes—that is, all but transparent mode—the user program should call the SERVEMOUSE routine to determine the source of an interrupt as soon as it receives one.

---

#### ***Transparent Mode***

In this mode, a program must read screen holes to check for mouse movement. In reality, however, an interrupt routine in the Apple IIc firmware updates mouse position counters each time the mouse is moved, then returns control to the main program task.

This is the only mouse mode available to BASIC programs.

---

#### ***Movement Interrupt Mode***

On the Apple IIc, a signal called VBLINT can interrupt the processor whenever a video vertical blanking signal occurs. This provides for efficient program coordination of the mouse cursor with mouse movement.

In movement interrupt mode, the mouse firmware arms VBLINT whenever the mouse is moved at least one count in any direction. When VBLINT occurs, program control passes to the vector address contained at locations \$3FE and \$3FF; the interrupt handler can then update the cursor smoothly to its next screen position.

The receiving interrupt handler must first call SERVEMOUSE (Table 9-3) to see if the mouse caused the interrupt. It should then call READMOUSE to get mouse status and its current X-Y position. The routine can also change the mouse mode and position if desired.

The maximum amount of mouse movement that can occur between successive VBLINT interrupts is limited only by the distance someone can move a mouse in one sixtieth of a second.

Section 5.2.2 contains recommendations for using MouseText characters with a mouse.

---

### ***Button Interrupt Mode***

The Apple IIc mouse-button hardware location does not generate interrupts. However, a program can simulate mouse-button interrupts by polling the button whenever VBLINT occurs, and acting on the interrupt whenever the button state has changed. This alleviates the program overhead required to poll the button constantly to provide fast response.

---

### ***Movement/Button Interrupt Mode***

This is a combination of the two modes just described. It provides the best response possible without constant polling of the mouse position and button. Processing of a main task can be concurrent with cursor and menu updating, as well as menu-selected command processing.

---

### ***Vertical Blanking Active Modes***

These modes are the same as the four just described except that they allow VBLINT interrupts to be sent to the user.

---

## ***9.1.3 Mouse Hardware Page Locations***

The soft switches assigned to the mouse interface are shown in Table 9-2. On power-up or reset, the hardware selects the rising edge of X0 and Y0 and masks out all mouse interrupts.

Mouse firmware sets interrupts in response to mode settings under program control. The vertical blanking interrupt (VBLINT) is armed if the mouse button is pushed or there is a change of at least a count of 1 in the X0 or Y0 coordinate. Since VBL occurs every sixtieth of a second, at most that amount of time will elapse before the resulting interrupt can be acknowledged and acted upon. To reset the VBL interrupt, read \$C070.

Software can also select which edge of X0 and Y0 information will cause the XINT or YINT.

Once an interrupt has occurred, you can read the mouse's X1 and Y1 direction in data-bus bit 7 by reading address \$C066 and \$C067, respectively.

A program can read the status of the soft switches by reading one of the locations \$C040-\$C043 and then testing data bit 7.

Section 11.12 explains what X0, Y0, X1, Y1 are and what they mean with respect to mouse movement.

Appendix E explains how the firmware handles interrupts.




---

**Warning**

*Table 9-2 is included here for your information; however, you should use the built-in firmware to access the mouse. If you do write your own mouse interrupt handler, it should enable the main bank-switched memory, set up its own IRQ vectors at addresses \$FFFE and \$FFFF, keep track of video modes and the alternate stack, and check for the interrupt source in the same manner as the mouse firmware listed in Volume 2 of this manual.*

*Using the built-in firmware is much easier and guarantees compatibility with all other Apple II series computers.*

---

**Table 9-2. Mouse Hardware Page Locations.** (1) When IOUDIS is on, \$C058-\$C05F do not affect mouse, and \$C05E and \$C05F become DHIRES (Table 5-8). (2) Read or write to \$C07x also resets VBLINT and triggers paddle timers. (3) These work only if IOUDIS is off. (4) This location is also the  key (Table 4-1). (5) This is also the location of the shift-key mod (Appendix F).

Action	Hex	Name	Function	Notes
W	\$C07E	IOUDIS	On: disable IOU access for addresses \$C058 to \$C05F; enable access to DHIRES switch	(1)
W	\$C07F	IOUDIS	Off: enable IOU access for addresses \$C058 to \$C05F; disable access to DHIRES switch	(1)
R7	\$C07E	RIOUDIS	Read IOUDIS switch (1 = off)	(2)
R/W	\$C058	DISXY	Disable (mask) X0 and Y0 movement interrupts	(3)
R/W	\$C059	ENBXY	Enable (allow) X0 and Y0 movement interrupts	(3)
R7	\$C040	RDXYMSK	Read status of X0/Y0 interrupt mask (1 = mask on)	
R	\$C048	RSTXY	Reset X0/Y0 interrupt flags	
R/W	\$C05C	X0EDGE	Select rising edge of X0 for interrupt	(3)
R/W	\$C05D	X0EDGE	Select falling edge of X0 for interrupt	(3)
R7	\$C042	RDX0EDGE	Read status of X0 edge selector (1 = falling)	
R	\$C015	RSTXINT	Reset mouse X0 interrupt flag	
R/W	\$C05E	Y0EDGE	Select rising edge of Y0 for interrupt	(3)
R/W	\$C05F	Y0EDGE	Select falling edge of Y0 for interrupt	(3)
R7	\$C043	RDY0EDGE	Read status of Y0 edge selector (1 = falling)	
R	\$C017	RSTYINT	Reset mouse Y0 interrupt flag	
R/W	\$C05A	DISVBL	Disable (mask) VBL interrupts	(3)
R/W	\$C05B	ENVBL	Enable (allow) VBL interrupts	(3)
R7	\$C041	RDVBLMSK	Read status of VBL interrupt mask (1 = mask on)	
R	\$C019	RSTVBL	Read and then reset VBLINT flag	
R/W	\$C070	PTRIG	Reset VBLINT flag; trigger paddle timer	

*Table 9-2—Continued. Mouse Hardware Page Locations*

Action	Hex	Name	Function	Notes
R7	\$C061	RDBTN0	Read hand control button status (1 = pressed)	(4)
R7	\$C063	RD63	Read mouse button status (0 = pressed)	(5)
R7	\$C066	MOUX1	Read status of X1 (mouse X direction) (1 = high)	
R7	\$C067	MOUY1	Read status of Y1 (mouse Y direction) (1 = high)	

### **9.1.4 I/O Firmware Support**

The Apple IIc supports the mouse with firmware starting at address \$C400. This firmware is necessary because the mouse requires fast, transparent interrupt processing to work effectively.

In assembly language, which you might need to use for sophisticated mouse applications, you can use direct firmware support. To enable the mouse, first load a mode byte into the accumulator (and \$C4 in X, \$40 in Y), and then do a JSR to the firmware routine called SETMOUSE (Table 9-3). Valid mode bytes are:

\$00	Turn mouse off
\$01	Set transparent mode
\$03	Set movement-interrupt mode
\$05	Set button-interrupt mode
\$07	Set movement- or button-interrupt mode
\$08	Turn mouse off, VBLINT active
\$09	Set transparent mode, VBLINT active
\$0B	Set movement-interrupt mode, VBLINT active
\$0D	Set button-interrupt mode, VBLINT active
\$0F	Set movement- or button-interrupt mode, VBLINT active

The firmware will then initialize the mouse. To read the current position and status of the mouse, first load \$C4 into the X register, load \$40 into the Y register, save processor status, disable interrupts, and then JSR to the firmware routine called READMOUSE (Table 9-3), which stores the information in the port 4 screen holes (Table 9-5).

Table 9-3 lists the mouse port firmware routine offsets. Each address contains the low byte of the entry point of the routine described. The calling setup for all routines (except SERVEMOUSE) is the same: the X register must contain \$C4, and the Y register must contain \$40. When the routine has finished, the A, X, and Y register contents are undefined.



**Table 9-3. Mouse Firmware Routines**

Location	Offset For	Description
\$C412	SETMOUSE	Sets the mouse mode to the value in the accumulator Input: A register contains mode (see \$7FC, Table 9-5). Output: Carry bit = 0 means mode was legal; carry bit = 1 means mode was not legal.
\$C413	SERVEMOUSE	Serves mouse interrupt if needed Input: X, Y, A registers—doesn't matter Output: Carry bit = 0 means mouse caused the interrupt; carry bit = 1 means something else caused it. This routine updates \$77C to show which event caused the interrupt (values in Table 9-5).
\$C414	READMOUSE	Updates screen holes to show current mouse X-Y position and button status; clears VBLINT, button and movement interrupt bits in the status byte. Don't re-enable interrupts until after retrieving position values. Output: Carry bit = 0
\$C415	CLEARMOUSE	Sets the mouse position to 0, though not necessarily within clamping boundaries; leaves button and interrupt bits in status byte unchanged. Output: Carry bit = 0
\$C416	POSMOUSE	Sets the mouse coordinates to new values. Input: X and Y screen holes contain new X and Y positions. Output: Carry bit = 0
\$C417	CLAMPMOUSE	Sets new clamping boundaries (see Table 9-5). Does not affect mouse position or update mouse position screen holes; use READMOUSE to do that. Input: A register = 0 means set new X boundaries; A register = 1 means set new Y boundaries. Output: Carry bit = 0
\$C418	HOMEMOUSE	Sets the internal mouse position to the upper-left corner of the clamping window. Does not update mouse position screen holes; use READMOUSE to do that.



*Table 9-3—Continued. Mouse Firmware Routines*

\$C419	INITMOUSE	Sets startup internal values; does not update mouse-position screen holes. Output: Carry bit = 0
--------	-----------	---

Here is a sample sequence of events and calls.

1. Four screen holes contain the mouse's X and Y coordinates, and one contains the status of the last mouse movement (Table 9-5).
2. Call INITMOUSE.
3. Inhibit interrupts, set up the boundaries you want, then call CLAMPMOUSE.
4. Use POSMOUSE, HOMEMOUSE or CLEARMOUSE to position the mouse where you want it.
5. Put the mode (see address \$7FC in Table 9-5) in the accumulator, then call SETMOUSE.
6. If you have set one of the interrupt modes, then when an interrupt arrives, call SERVEMOUSE to determine the source of the interrupt.
7. Disable interrupts and call READMOUSE. Retrieve the position values, then re-enable interrupts.

---

### ***Pascal Support***

Table 9-4 lists the locations and values of the I/O firmware protocol that Pascal 1.1 and 1.2 use. However, Pascal must use a special attach driver to support the mouse.

***Table 9-4. Mouse Port I/O Firmware Protocol***

<b>Address</b>	<b>Value</b>	<b>Description</b>
\$C405	\$38	Pascal ID byte
\$C407	\$18	Pascal ID byte
\$C40B	\$01	Generic signature byte of firmware cards
\$C40C	\$20	2 = X-Y pointing device; 0 = identification code
\$C40D		Initialization routine (not implemented; returns error code)
\$C40E		Standard read routine (not implemented; returns error code)
\$C40F		Standard write routine (not implemented; returns error code)
\$C410		Standard status routine (not implemented; returns error code)
\$C411	\$00	Optional routines follow.
\$C4FB	\$D6	A mouse identification byte

---

### ***BASIC and Assembly-Language Support***

In BASIC, before you can get input from the mouse, you must turn it on by printing PR#4 and then CHR\$(1). This sets transparent mode. After that, re-enable video output with PR#3, and take subsequent input from the mouse by issuing IN#4. The first input statement after that (INPUT X,Y,S) initializes and enables the mouse, and returns a three-element string:

+xxx,+yyy,+st

representing the x-coordinate, y-coordinate and status digits.

The coordinates will be integers between 0 and +1023. These are called the **clamping boundaries** of the mouse.

The sign preceding the status digits is normally positive; it becomes negative when you press a key on the keyboard.

The first digit, *s*, of the status is 0. The second digit, *t*, of the status is 1 if the mouse button is still pressed, 2 if it was just pressed, 3 if it was just released, and 4 if it is still released.

To disable the mouse:

```
PRINT CHR$(4)"PR#4"  
PRINT CHR$(0)  
PRINT CHR$(4)"PR#3"
```

---

### ***9.1.5 Screen Holes***

Table 9-5 lists the screen holes that the mouse firmware uses. Note that the mouse firmware reserves port 5 screen holes for its own use. Also, the auxiliary-page counterparts of the port 4 addresses are reserved for startup values.

**Note:** Some screen holes are different for the Apple IIe mouse. Refer to Appendix F.

**Table 9-5. Mouse Peripheral Card RAM Locations**

Scratch Area:																			
Location	Description																		
\$478	Low byte of clamping minimum																		
\$4F8	Low byte of clamping maximum																		
\$578	High byte of clamping minimum																		
\$5F8	High byte of clamping maximum																		
Port 4 Screen Holes:																			
Location	Description																		
\$47C	Low byte of X coordinate																		
\$4FC	Low byte of Y coordinate																		
\$57C	High byte of X coordinate																		
\$5FC	High byte of Y coordinate																		
\$67C	Reserved																		
\$6FC	Reserved																		
\$77C	Status byte																		
	<table><tr><th>Bit</th><th>1 Equals</th></tr><tr><td>7</td><td>Button down</td></tr><tr><td>6</td><td>Button was down on last read and still down.</td></tr><tr><td>5</td><td>Movement since last read</td></tr><tr><td>4</td><td>Reserved</td></tr><tr><td>3</td><td>Interrupt from VBLINT</td></tr><tr><td>2</td><td>Interrupt from button</td></tr><tr><td>1</td><td>Interrupt from movement</td></tr><tr><td>0</td><td>Reserved</td></tr></table>	Bit	1 Equals	7	Button down	6	Button was down on last read and still down.	5	Movement since last read	4	Reserved	3	Interrupt from VBLINT	2	Interrupt from button	1	Interrupt from movement	0	Reserved
Bit	1 Equals																		
7	Button down																		
6	Button was down on last read and still down.																		
5	Movement since last read																		
4	Reserved																		
3	Interrupt from VBLINT																		
2	Interrupt from button																		
1	Interrupt from movement																		
0	Reserved																		
\$7FC	Mode byte (current mode; mask out bits 4-7 when testing)																		
	<table><tr><th>Bit</th><th>1 Equals</th></tr><tr><td>7-4</td><td>Reserved</td></tr><tr><td>3</td><td>VBLINT active</td></tr><tr><td>2</td><td>VBLINT interrupt on button</td></tr><tr><td>1</td><td>VBLINT interrupt on movement</td></tr><tr><td>0</td><td>Mouse active</td></tr></table>	Bit	1 Equals	7-4	Reserved	3	VBLINT active	2	VBLINT interrupt on button	1	VBLINT interrupt on movement	0	Mouse active						
Bit	1 Equals																		
7-4	Reserved																		
3	VBLINT active																		
2	VBLINT interrupt on button																		
1	VBLINT interrupt on movement																		
0	Mouse active																		
Port 5 Screen Holes:																			
	Reserved																		

---

### 9.1.6 Using the Mouse as a Hand Control

This section describes how to use the mouse as if it were a set of hand controls, or an X-Y pointing device in port 4. If you turn the mouse on, the Monitor hand-control (game paddle) routines will take input from the mouse. This is possible because the mouse and the hand controls all use the same back-panel connector.

You can run a BASIC program that uses the PDL function to read from the mouse by doing this:

1. Start up the system with the BASIC program that uses paddles.
2. Type PR#4 and press **(RETURN)** to turn on the mouse.
3. Press **(CONTROL)-(A)** **(RETURN)** to initialize the mouse.
4. Type PR#0 and press **(RETURN)** to restore output to the screen.
5. RUN the program.

Play the game using the mouse instead of the paddles.

**Note:** Many copy-protected games will not work with a mouse. Also, many games don't use built-in firmware for the paddles.

## 9.2 Game Input

The Apple IIc supports game paddles, joysticks, and other hand controls connected to the DB-9 connector on its back panel. Table 9-6 is a summary of game input characteristics.

*Table 9-6. Game Input Characteristics*

Port Number	None
Commands	None
Initial Characteristics	Game inputs cannot be disabled.
Hardware Page Locations	Description
\$C061	Switch input 0 and (↩)
\$C062	Switch input 1 and (⬇)
\$C063	Mouse button. (Sense is opposite that of \$C061 to distinguish it from paddle button)
\$C064	Analog input (paddle) 0
\$C065	Analog input (paddle) 1
\$C070	Trigger paddle timer
Monitor Firmware Routines	Name      Description
\$FB1E	PREAD      Read a paddle position
I/O Firmware Entry Points	None
Use of Screen Notes	None

### 9.2.1 The Hand Control Connector Signals

Several inputs are available on a 9-pin D-type miniature connector on the back of the Apple IIc: two one-bit inputs, or switches, and two analog inputs. You can access all of these signals from your programs.

When you connect a pair of hand controls to the 9-pin connector, the rotary controls use two analog inputs, and the push-buttons use two one-bit inputs. However, you can also use these inputs for many other jobs. For example, two analog inputs can be used with a two-axis joystick.




Complete electrical specifications of these inputs are given in Chapter 11; Table 11-22 shows the connector pin numbers.



---

### *Switch Inputs (SW0 and SW1)*

The two one-bit inputs can be connected to the output of another electronic device that meets the electrical requirements (Chapter 11), or to a pushbutton. When you read a byte from one of these locations, only the high-order bit—bit 7—is valid information; the rest of the byte is undefined. From machine language, you can do a Branch Plus or Branch Minus on the state of bit 7. From BASIC, you read the switch with a PEEK and compare the value with 128. If the value is 128 or greater, the switch is on.

The memory locations for these switches are \$C061, \$C062 and \$C063, as shown in Table 9-6. Switch 0 and switch 1 are permanently connected to  and  on the keyboard; these are the ones connected to the buttons on the hand controls. Location \$C063 is a second address for the mouse button, so that a program can distinguish it from an  keypress. When the mouse button is pressed, \$C063 (Bit 7) goes from 1 to 0, and \$C061 (Bit 7) goes from 0 to 1. When the mouse button is pressed, \$C063 (Bit 7) goes from 1 to 0.

---

### ***Analog Inputs (PDL0 and PDL1)***

The two analog inputs are designed for use with 150K ohm variable resistors or potentiometers. The variable resistance is connected between the +5V supply and each input, so that it makes up part of a timing circuit (refer to section 11.13 for details). The circuit changes state when its time constant has elapsed, and the time constant varies as the resistance varies. Your program can measure this time by counting in a loop until the circuit changes state, or times out.

Before a program can read the analog inputs, it must first reset the timing circuits. Accessing memory location \$C070 does this. As soon as you reset the timing circuits, the high bits of the bytes at locations \$C064 through \$C067 are set to one. If you PEEK at them from BASIC, the values will be 128 or greater. Within about 3 milliseconds, these bits will change back to zero—byte values less than 128—and remain there until you reset the timing circuits again. The exact time each of the bits remains high is directly proportional to the resistance connected to the corresponding input. If these inputs are open—no resistances are connected—the corresponding bits may remain high indefinitely.

You can read and reread the same paddle at arbitrarily short intervals. However, you must wait at least 3 milliseconds between reading one paddle and reading a different paddle.

---

### 9.2.2 Monitor Support

To read the analog inputs from machine language, you can use a program loop that resets the timers and then increments a counter until the bit at the appropriate memory location changes to zero, or you can use the built-in routine `PREAD`. BASIC and other high-level languages also include convenient means of reading the analog inputs—refer to your language manuals.

---

#### **`PREAD`**

The Monitor routine `PREAD` (at address `$FB1E`) places in the Y register a number between `$00` and `$FF` that represents the position of a hand control. You pass the number of the hand control in the X register.



---

#### **Warning**

*If the hand control number you furnish in the X register does not equal 0 or 1, strange things may happen.*



---

#### **Warning**

*The paddle and vertical blanking both use `$C070`. If you are reading the paddles and using VBL interrupts, disable interrupts before calling `PREAD`.*

---

# *Using the Monitor*

The System Monitor is a set of subroutines in the Apple IIc firmware. The Monitor provides a standard interface to the built-in I/O devices described in Chapter 1. Many of the I/O subroutines described in Chapters 3 through 9 are part of the System Monitor.

DOS (but not ProDOS) and the BASIC interpreters (Appendix E) use these subroutines by direct calls to their starting locations. The starting addresses for all of the standard subroutines are listed in Appendix C. If you wish, you can call the standard subroutines from your programs in the same fashion.

You can perform most of the Monitor functions directly from the keyboard. This chapter tells you how to use the Monitor

- to look at one or more memory locations
- to change the contents of any location
- to write programs in machine language to be executed directly by the Apple IIc's microprocessor
- to move and compare blocks of memory
- to invoke other programs from the Monitor.

---

## 10.1 Invoking the Monitor

The positive and negative decimal equivalents of Monitor locations are listed in Appendix C. In addition, Appendix H contains conversion tables from one numbering system to another.

The System Monitor starts at memory location \$FF69 (-151). To invoke the Monitor, you make a CALL statement to this location from the keyboard or from a BASIC program. When the Monitor is running, its prompting character, an asterisk (\*), appears on the left side of the display screen, followed by a cursor.

To use the Monitor, you type commands at the keyboard. When you have finished using the Monitor, you return to the BASIC language you were previously using by pressing **CONTROL-RESET**, by pressing **CONTROL-C** and then **RETURN**, or by typing **3D0G**, which executes the resident program—usually Applesoft—whose address is stored in a jump instruction at location **\$3D0**.

**Note:** If DOS or ProDOS is connected via the standard I/O links (Chapter 3), then you can issue commands to it from the Monitor. Under this arrangement, errors will return control to BASIC rather than to the Monitor.

If you want to have **CONTROL-RESET** return you to the Monitor, load the values **\$69**, **\$FF**, and **\$5A** into the three locations starting at address **\$3F2** (the reset-vector address and the power-up byte).

---

## 10.2 Syntax of Monitor Commands

To give a command to the Monitor, you type a line on the keyboard, then press **RETURN**. The Monitor accepts the line using the standard I/O subroutine GETLN described in Chapter 3. A Monitor command can be up to 255 characters in length, ending with a carriage return.

A Monitor command can include three kinds of information: addresses, data values, and command characters. You type addresses and data values in hexadecimal notation.

When the command you type calls for an address, the Monitor accepts any group of hexadecimal digits. If there are fewer than four digits in the group, it adds leading zeros; if there are more than four hexadecimal digits, the Monitor uses only the last four digits. It follows a similar procedure when the command syntax calls for two-digit data values.

Each command you type consists of one command character, usually the first letter of the command name. The Monitor recognizes 22 different command characters. Some of them are punctuation marks, some are letters (uppercase or lowercase), and some are control characters.



**Note:** Although the Monitor recognizes and interprets them, control characters typed on an input line do not appear on the screen.

This chapter contains many examples of the use of Monitor commands. Some of the data values displayed by your Apple IIc may differ from the values printed in these examples, because they are variables stored in programmable memory.

## 10.3 Monitor Memory Commands

When you use the Monitor to examine and change the contents of memory, it keeps track of the address of the last location whose value you inquired about and the address of the location that is next to have its value changed. These are called the **last opened location** and the **next changeable location**.

### Warning

*Because locations \$C000 through \$C0FF contain special hardware circuits, issuing any command that reads or writes on this page can have unpredictable, and perhaps disastrous, results.*

### 10.3.1 Examining Memory Contents

When you type the address of a memory location and press **(RETURN)**, the Monitor responds with the address you typed, a dash, a space, and the value stored at that location, like this:

\*E000

E000- 4C

\*33

0033- AA

\*

Each time the Monitor displays the value stored at a location, it saves that address as the last opened location and as the next changeable location.

---

### 10.3.2 Memory Dump

When you type a period (.) followed by an address, and then press **RETURN**, the Monitor displays a **memory dump**: the data values stored at all the memory locations from the one following the last opened location to the location whose address you typed following the period. The Monitor saves the last location displayed as both the last opened location and the next changeable location. In these examples, the amount of data displayed by the Monitor depends on how much larger the address after the period is than the last opened location.

\*20

0020- 00

\*.2B

0021- 28 00 18 0F 0C 00 00

0028- A8 06 D0 07

\*300

0300- 99

\*.315

0301- B9 00 08 0A 0A 0A 99

0308- 00 08 C8 D0 F4 A6 2B A9

0310- 09 85 27 AD CC 03

\*.32A

0316- 85 41

0318- 84 40 8A 4A 4A 4A 09

0320- C0 85 3F A9 5D 85 3E 20

0328- 43 03 20

\*

When the Monitor performs a memory dump, it starts at the location immediately following the last opened location and displays that address and the data value stored there. It then displays the values of successive locations up to and including the location whose address you typed, but only up to eight values on a line. When it reaches a location whose address is a

multiple of eight—that is, one that ends with an 8 or a 0—it displays that address as the beginning of a new line, then continues displaying more values.

After the Monitor has displayed the value at the location whose address you specified in the command, it stops the memory dump and sets that location as both the last opened location and the next changeable location. If the address specified on the input line is less than the address of the last opened location, the Monitor displays only the address and value of the location following the last opened location.

You can combine the two commands, opening a location and dumping memory, by simply concatenating them: type the first address, a period, and the second address. This combination of two addresses separated by a period is called a **memory range**.

\*300.32F

```
0300- 99 B9 00 08 0A 0A 0A 99
0308- 00 08 08 D0 F4 A6 2B A9
0310- 09 85 27 AD CC 03 85 41
0318- 84 40 8A 4A 4A 4A 4A 09
0320- C0 85 3F A9 5D 85 3E 20
0328- 43 03 20 46 03 A5 3D 4D
*30.40
```

```
0030- AA 00 FF AA 05 C2 05 C2
0038- 1B FD D0 03 3C 00 40 00
0040- 30
*E015.E025
```

```
E015- 4C ED FD
E01B- A9 20 C5 24 B0 0C A9 8D
E020- A0 07 20 ED FD A9
*
```

Pressing **(RETURN)** by itself causes the Monitor to display one line of a memory dump; that is, a memory dump from the location following the last opened location to the next multiple-of-eight boundary. The Monitor saves the address of the last location displayed as both the last opened location and the next changeable location.

\*5

0005- 00

\* RETURN

00 00

\* RETURN

0008- 00 00 00 00 00 00 00 00

\*32

0032- FF

\* RETURN

AA 00 C2 05 C2

\* RETURN

0038- 1B FD D0 03 3C 00 3F 00

\*

---

### 10.3.3 Changing Memory Contents

Section 10.3.2 showed you how to display values stored in the Apple IIc's memory; this section shows you how to change these values. You can change any location in RAM; you can change the characteristics and treatment of an output device by changing the contents of locations assigned to it; and you can change a soft switch setting by referencing its set and reset addresses.



---

#### Warning

*Use these commands carefully. If you change the zero-page locations used by the interpreter or operating system (Appendix B), you may lose programs or data stored in memory.*

---

---

#### Changing One Byte

The previous commands keep track of the next changeable location; these commands make use of it. In the next example, you open location 0, then type a colon followed by a value.

\*0

0000- 4C

\*:5F

The contents of the next changeable location have just been changed to the value you typed, as you can see by examining that location:

\*0

0000- 5F

\*

You can also combine opening and changing into one operation by typing an address followed by a colon and a value. In the example, you type the address again to verify the change.

\*302:42

\*302

0302- 42

\*

When you change the contents of a location, the value that was contained in that location is replaced by the new value, which will remain until you replace it with another value.

---

### ***Changing Consecutive Locations***

You don't have to type a separate command with an address, a colon, a value, and press **(RETURN)** for each location you want to change. You can change the values of up to eighty-five consecutive locations at a time—or even more, if you omit leading zeros from the values—by typing only the initial address and colon followed by all the values separated by spaces; end with **(RETURN)**. The Monitor will store the consecutive values in consecutive locations, starting at the location whose address you typed. After it has processed the string of values, it takes the location following the last changed location as the next

changeable location. Thus, you can continue changing consecutive locations, without typing an address on the next input line, by typing another colon and more values. In these examples, you first change some locations, then examine them to verify the changes.

```
*300:69 01 20 ED FD 4C 0 3
```

```
*300
```

```
0300- 69
```

```
* (RETURN)
```

```
01 20 ED FD 4C 00 03
```

```
*10:0 1 2 3
```

```
*:4 5 6 7
```

```
*10.17
```

```
0010- 00 01 02 03 04 05 06 07
```

```
*
```

---

### 10.3.4 Moving Data in Memory

You can copy a block of data stored in a range of memory locations from one area in memory to another by using the Monitor's MOVE command. To move a range of memory, you must tell the Monitor both where the data is now situated in memory—the source locations—and where you want the copy to go—the destination locations. You give this information to the Monitor by means of three addresses: the address of the first location in the destination and the addresses of the first and last locations in the source. You specify the starting and ending addresses of the source range by separating them with a period. You separate the destination address from the range addresses with a less-than character (<), which you may think of as an arrow pointing in the direction of the move. Finally, you tell the Monitor that this is a MOVE command by pressing (M). The format of the complete MOVE command looks like this:

```
|destination| < |start| . |end| (M)
```



When you type the actual command, replace the words in braces with hexadecimal addresses, and omit the braces and spaces. Here are some examples of memory moves. First, you examine the values stored in one range of memory, then store several values in another range of memory. The actual MOVE commands end with (M).

\*0.F

```
0000- 5F 00 05 07 00 00 00 00
0008- 00 00 00 00 00 00 00 00
*300:A9 8D 20 ED FD A9 45 20 DA FD 4C 00
03
```

\*300.30C

```
0300- A9 8D 20 ED FD A9 45 20
0308- DA FD 4C 00 03
*0<300.30C(M)
```

\*0.C

```
0000- A9 8D 20 ED FD A9 45 20
0008- DA FD 4C 00 03
*310<8.A(M)
```

\*310.312

```
0310- DA FD 4C
*2<7.9(M)
```

\*0.C

```
0000- A9 8D 20 DA FD A9 45 20
0008- DA FD 4C 00 03
*
```

The Monitor moves a copy of the data stored in the source range of locations to the destination locations. The values in the source range are left undisturbed. The Monitor remembers the

See section 10.6 for an interesting application of this feature.

last location in the source range as the last opened location, and the first location in the source range as the next changeable location. If the second address in the source range specification is less than the first, then only one value (that of the first location in the range) will be moved.

If the destination address of the MOVE command is inside the source range of addresses, then strange things happen: the locations between the beginning of the source range and the destination address are treated as a sub-range and the values in this sub-range are replicated throughout the source range.

---

### ***10.3.5 Comparing Data in Memory***

You can use the VERIFY command to compare two ranges of memory using the same format you use to move a range of memory from one place to another. In fact, the VERIFY command can be used immediately after a MOVE to make sure that the move was successful.

The VERIFY command, like the MOVE command, needs a range and a destination. The syntax of the VERIFY command is:

|destination| < |start| . |end| (V)

The Monitor compares the values in the source locations with the values in the locations beginning at the destination address. If any values don't match, the Monitor displays the address at which the discrepancy was found and the two values that differ. In the example, you store data values in the range of locations from 0 to \$D, copy them to locations starting at \$300 with the MOVE command, and then compare them using the VERIFY command. When you use the VERIFY command after you change the value at location 6 to \$E4, it detects the change.

\*0:D7 F2 E9 F4 F4 E5 EE A0 E2 F9 A0 C3 C4  
C5

\*300<0.D(M)

\*300<0.D(V)

\*6:E4

\*300<0.D(V)

0006-E4 (EE)

\*

Like the MOVE command, the VERIFY command also does unusual things if the destination address is within the source range; see section 10.6.

If the VERIFY command finds a discrepancy, it displays the address of the location in the source range whose value differs from its counterpart in the destination range. If there is no discrepancy, VERIFY displays nothing. The VERIFY command leaves the values in both ranges unchanged. The last opened location is the last location in the source range, and the next changeable location is the first location in the source range, just as in the MOVE command. If the ending address of the range is less than the starting address, the values of only the first locations in the ranges will be compared.

## 10.4 Monitor Register Commands

Even though the actual contents of the 65C02's internal registers are changing as you use the Monitor, you can examine the values that the registers contained at the time the Monitor gained control, either because you called it or because the program you are debugging stopped at a break (BRK). You can also store new register values that will be used when you execute a program from the Monitor using the GO command, described below.

---

### 10.4.1 Changing Registers

When you call the Monitor, it stores the contents of the 65C02 registers in memory. The registers are stored in the order A, X, Y, P (processor status register), and S (stack pointer), starting at location \$45. When you give the Monitor a GO command, the Monitor loads the registers from these five locations before it executes the first instruction in your program.

---

### 10.4.2 Examining Registers

Pressing **(CONTROL)-(E)** and then **(RETURN)** invokes the Monitor's EXAMINE command, which displays the stored register values and sets the location containing the contents of the A register as the next changeable location. After using the EXAMINE command, you can change the values in these locations by typing a colon and then typing the new values separated by spaces. In the following example, you display the registers, change the first two, and then display them again to verify the change.

\* **(CONTROL)-(E)**

A=0A X=FF Y=D8 P=B0 S=F8  
\*:B0 02

\* **(CONTROL)-(E)**

A=B0 X=02 Y=D8 P=B0 S=F8  
\*

## 10.5 Miscellaneous Monitor Commands

These Monitor commands enable you to change the video display format from normal to inverse and back, and to assign input and output to external devices.

### 10.5.1 Display Inverse and Normal

The COUT subroutine is described in Chapter 3.

You can control the setting of the inverse-normal mask location used by the COUT subroutine from the Monitor so that all the Monitor's output will be in inverse format. The INVERSE command I sets the mask such that all subsequent inputs and outputs are displayed in inverse format. To switch the Monitor's output back to normal format, use the NORMAL command N.

\*0.F

```
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
```

\*(I)

\*0.F

```
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
```

\*(N)

\*0.F

```
0000- 0A 0B 0C 0D 0E 0F D0 04
0008- C6 01 F0 08 CA D0 F6 A6
```

\*

See Appendix D.

---

### 10.5.2 Back to BASIC

If you are using one of the Apple disk operating systems (ProDOS or DOS), press **(CONTROL)-(RESET)** or type

3D0G

to return to the language you were using, with your program and variables intact.

**Note:** If you type the latter command, make sure that the third character you type is a *zero*, not a letter *O*. The letter *G* is the Monitor's GO command, described below in section 10.7.

If there is no operating system in RAM, use the BASIC command **(CONTROL)-(B)** to leave the Monitor and enter the BASIC interpreter that was active when you entered the Monitor. (Normally this is Applesoft BASIC.) Any program or variables that you had previously in BASIC will be lost. If you want to re-enter BASIC with your previous program and variables intact, use the CONTINUE BASIC command **(CONTROL)-(C)**.



Chapter 3 lists the Apple IIc port numbers available.

For more information on the way those commands work, refer to section 3.1.

---

### 10.5.3 Redirecting Input and Output

The CONTROL-P command diverts all output normally destined for the screen (port 0) to a device attached to one of the other ports, from 1 to 7. The format of the command is

|port number| (CONTROL)-(P)

A CONTROL-P command to port number 0 will switch the stream of output characters back to the Apple IIc's video display. However, use (ESC) (CONTROL)-(Q) if the enhanced video firmware is active (solid-block cursor).

In much the same way that the CONTROL-P command switches the output stream, the CONTROL-K command substitutes a device connected to a specified port for the Apple IIc's normal input device, the keyboard. The format for the command is:

|port number| (CONTROL)-(K)

Pressing (D) (CONTROL)-(K) directs the Monitor to accept input from the Apple IIc's built-in keyboard.

The CONTROL-P and CONTROL-K commands are the exact equivalents of the BASIC (but not DOS and ProDOS) commands PR# and IN#.

---

### 10.5.4 Hexadecimal Arithmetic

The Monitor will also perform one-byte hexadecimal addition and subtraction. Just type a line in one of these formats followed by (RETURN)

|value| + |value| (RETURN)  
|value| - |value| (RETURN)

The Apple IIc performs the arithmetic and displays the result, as shown in these examples.

```
*20+13
=33
*4A-C
=3E
*FF+4
=03
*3-4
=FF
*
```

## 10.6 Special Tricks With the Monitor

This section describes some more complex ways of using the Monitor commands.

### 10.6.1 Multiple Command Lines

You can put as many Monitor commands on a single line as you like, as long as you separate them with spaces and the total number of characters in the line is less than 254. Adjacent single-letter commands such as **(L)**, **(S)**, **(I)**, and **(N)** need not be separated by spaces.

You can freely intermix all of the commands except the **STORE (:** command. Since the Monitor takes all values following a colon and places them in consecutive memory locations, the last value in a **STORE** must be followed by a letter command before another address is encountered. You can use the **NORMAL** command as the required letter command in such cases; it usually has no effect and can be used anywhere.

In the following example, you display a range of memory, change it, and display it again, all with one line of commands.

```
*300.307 300:18 69 1 (N) 300.302
```

```
0300- 00 00 00 00 00 00 00 00
0300- 18 69 01
*
```

If the Monitor encounters a character in the input line that it does not recognize as either a hexadecimal digit or a valid command character, it executes all the commands on the input line up to that character, then grinds to a halt with a noisy beep and ignores the remainder of the input line.

---

### 10.6.2 Filling Memory

The MOVE command can be used to replicate a pattern of values throughout a range of memory. To do this, first store the pattern in the first locations in the range:

```
*300:11 22 33
```

•

Remember the number of values in the pattern: in this case, it is 3. Use the number to compute addresses for the MOVE command, like this:

```
|start+number| < |start| , |end-number| (M)
```

This MOVE command will first replicate the pattern at the locations immediately following the original pattern, then replicate that pattern following itself, and so on until it fills the entire range.

```
*303<300.32D(M)
```

```
*300.32F
```

```
0300- 11 22 33 11 22 33 11 22
0308- 33 11 22 33 11 22 33 11
0310- 22 33 11 22 33 11 22 33
0318- 11 22 33 11 22 33 11 22
0320- 33 11 22 33 11 22 33 11
0328- 22 33 11 22 33 11 22 33
•
```

You can do a similar trick with the VERIFY command to check whether a pattern repeats itself through memory. This is especially useful to verify that a given range of memory locations all contain the same value. In this example, to see the VERIFY command detect the discrepancy, you first fill the memory range from \$300 to \$320 with zeros and verify it, then change one location and verify again:

```
*300:0
```

```
*301<300.31F(M)
```

```
*301<300.31F(V)
```

```
*304:02
```

```
*301<300.31F(V)
```

```
0303-00 (02)
```

```
0304-02 (00)
```

```
*
```

---

### 10.6.3 Repeating Commands

You can create a command line that repeats one or more commands over and over. You do this by beginning the part of the command line that you want to repeat with a letter command, such as (N), and ending it with the sequence 34:n, where *n* is a hexadecimal number that specifies the position in the line of the command where you want to start repeating; for the first character in the line, *n*=0. The value for *n* must be followed with a space in order for the loop to work properly.

This trick takes advantage of the fact that the Monitor uses an index register to step through the input buffer, starting at location \$200. Each time the Monitor executes a command, it stores the value of the index at location \$34; when that command is finished, the Monitor reloads the index register with the value at location \$34. By making the last command change the value at location \$34, you change this index so that the Monitor picks up the next command character from an earlier point in the buffer.

The only way to stop a loop like this is to press **CONTROL-RESET**; that is how this example ends.

\* (N) 300 302 34:0 (N)

0300- 11  
0302- 33  
0300- 11  
0302- 33  
0300- 11  
0302- 33  
0300- 11  
0302- 33  
0300- 11  
0302- 33  
0300- 11  
0302- 33  
030

#### 10.6.4 Creating Your Own Commands

The **USER** command, **(CONTROL-Y)**, forces the Monitor to jump to memory location \$3F8. You can put a **JMP** instruction there that jumps to your own machine-language program. Your program can then examine the Monitor's registers and pointers or the input buffer itself to obtain its data. For example, here is a program that displays everything on the input line after the **(CONTROL-Y)**. The program starts at location \$300; the command line that starts with \$3F8 stores a jump to \$300 at location \$3F8.

```
*300:A4 34 B9 00 02 20 ED FD C8 C9 8D D0
F5 4C 69 FF
```

\*3F8:4C 00 03

```
* CONTROL-Y THIS IS A TEST
THIS IS A TEST
```

•

## 10.7 Machine-Language Programs

The main reason to program in machine language is to get more speed. A program in machine language can run much faster than the same program written in high-level languages such as BASIC or Pascal, but the machine-language version usually takes a lot longer to write. There are other reasons to use machine language: you might want your program to do something that isn't included in your high-level language, or you might just enjoy the challenge of using machine language to work directly on the bits and bytes.

**Note:** If you have never used machine language before, you'll need to learn the 65C02 instructions listed in Appendix A. To become proficient at programming in machine language, you'll have to spend some time at it, and study one of the books on 65C02 programming listed in the Bibliography.

You can get a hexadecimal dump of your program or move it around in memory using the commands described in the previous sections. The Monitor commands in this section are intended specifically for you to use in creating, writing, and debugging machine-language programs.

---

### 10.7.1 Running a Program

The Monitor command to start execution of your machine-language program is the GO command. When you type an address and press (G), the Apple IIc starts executing machine-language instructions starting at the specified location. If you just press (G), execution starts at the last opened location. The Monitor treats this program as a subroutine: it should end with an RTS (return from subroutine) instruction to transfer control back to the Monitor.

The Monitor has some special features that make it easier for you to write and debug machine-language programs, but before you get into that, here is a small machine-language program that you can run using only the simple Monitor commands already described. The program in the example merely displays the letters A through Z: you store it starting at location \$300, examine it to be sure you typed it correctly, then type 300G to start it running.



```
*300:A9 C1 20 ED FD 18 69 1 C9 DB D0 F6 60
```

```
*300.300
```

```
0300- A9 C1 20 ED FD 18 69 01
```

```
0308- C9 DB D0 F6 60
```

```
*300(G)
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
*
```

Since programs that translate assembly language into machine language are called **assemblers**, a program like the Monitor's LIST command that translates machine language into assembly language is called a **disassembler**.

The word **mnemonic** comes from the same root as memory and refers to short acronyms that are easier to remember than the hexadecimal operation codes themselves. For example, for *clear carry* you write CLC instead of \$18.

---

### 10.7.2 Disassembled Programs

Machine-language code in hexadecimal isn't the easiest thing in the world to read and understand. To make this job a little easier, machine-language programs are usually written in assembly language and converted into machine-language code by programs called **assemblers**.

The Monitor's LIST command displays machine-language code in assembly-language form. Instead of unformatted hexadecimal gibberish, the LIST command displays each instruction on a separate line, with a three-letter instruction name, or **mnemonic**, and a formatted hexadecimal **operand**. The LIST command also converts the relative addresses used in branch instructions to absolute addresses.

The Monitor LIST command has the format:

```
|location| (L)
```

The LIST command starts at the specified location and displays as much memory as it takes to make up a screenfull (20 lines) of instructions, as shown in the following example:

\*300(L)

0300-	A9 C1	LDA	#\$C1
0302-	20 ED	JSR	\$FDED
	FD		
0305-	18	CLC	
0306-	69 01	ADC	#\$01
0308-	C9 DB	CMP	#\$DB
030A-	D0 F6	BNE	\$0302
030C-	60	RTS	
030D-	00	BRK	
030E-	00	BRK	
030F-	00	BRK	
0310-	00	BRK	
0311-	00	BRK	
0312-	00	BRK	
0313-	00	BRK	
0314-	00	BRK	
0315-	00	BRK	
0316-	00	BRK	
0317-	00	BRK	

```

0318-    00          BRK
0319-    00          BRK
*
```

The first seven lines of this example are the assembly-language form of the program you typed in the previous example. The rest of the lines are BRK instructions only if this part of memory has zeros in it; other values will be disassembled as other instructions.

The Monitor saves the address that you specify in the LIST command, but not as the last opened location used by the other commands. Instead, the Monitor saves this address as the **program counter**, which it uses only to point to locations within programs. Whenever the Monitor performs a LIST command, it sets the program counter to point to the location immediately following the last location displayed on the screen, so that if you type another LIST command it will display another screenfull of instructions, starting where the previous display left off.

## 10.8 Summary of Monitor Commands

Here is a summary of the Monitor commands, showing the syntax diagram for each one.

### Examining Memory

<code>adrs (RETURN)</code>	Displays the value contained in one location.
<code>adrs1 , adrs2 (RETURN)</code>	Displays the values contained in all locations between 'adrs1' and 'adrs2'.
<code>(RETURN)</code>	Displays the values in up to eight locations following the last opened location.
<code>.adrs (L)</code>	Lists disassembled code starting at 'adrs', and continuing until the screen is full.

---

### *Changing the Contents of Memory*

<code>adrs[: val  val ...</code>	STORE command. Stores the values in consecutive memory locations starting at <code>adrs</code> .
<code>: val 'val ...</code>	Stores values in memory starting at the next changeable location.

---

### *Moving and Comparing*

<code> dest &lt; start .end  (M)</code>	MOVE command. Copies the values in the range <code> start .end </code> into the range beginning at <code> dest </code> .
<code> dest &lt;  start .end  (V)</code>	VERIFY command. Compares the values in the range <code> start .end </code> to those in the range beginning at <code> dest </code> .

---

### *The Register Command*

<code>(CONTROL)-(E)</code>	EXAMINE command. Displays the locations where the contents of the 65C02's registers are stored and opens them for changing.
----------------------------	---

---

### *Miscellaneous Monitor Commands*

<code>(I)</code>	INVERSE command. Sets inverse display mode.
<code>(N)</code>	NORMAL command. Sets normal display mode.
<code>(CONTROL)-(B)</code>	BASIC command. Enters the language currently active (normally Applesoft).
<code>(CONTROL)-(C)</code>	CONTINUE BASIC command. Returns to the language currently active (normally Applesoft).
<code>val +  val </code>	Adds the two values and prints the hexadecimal result.
<code> val  -  val </code>	Subtracts the second value from the first and prints the result.

|port| (CONTROL)-(P)

Redirects output to the device connected to port number |port|. If |port|=0, sends output to the video display. Use only when the enhanced video firmware is not active (checkerboard cursor).

(ESC) (CONTROL)-(Q)

Redirects output to video display when enhanced video firmware is active (solid block cursor).

|port| (CONTROL)-(K)

Takes input from the device connected to port number |port|. If |port|=0, accepts input from the keyboard.

(CONTROL)-(Y)

USER command. Jumps to the machine-language subroutine at location \$3F8.

---

### *Running and Listing Programs*

|adrs| (G)

Transfers control to the machine language program beginning at |adrs|.

|adrs| (L)

Disassembles and displays 20 instructions starting at |adrs|. Subsequent (L)'s display 20 more instructions each.

# *Hardware Implementation*



Most of this manual describes functions—what the Apple IIc does. This chapter, on the other hand, describes objects: the pieces of hardware the Apple IIc uses to carry out its functions. If you are designing a device to connect to the Apple IIc back panel, or if you just want to know more about how the Apple IIc is built, you should study this chapter.

## 11.1 Environmental Specifications

The Apple IIc is quite sturdy when used in the way it was intended—as a transportable computer, made for use in an indoor environment. You can carry it by its handle from room to room, but for longer trips Apple recommends that you use its carrying case or some other protective container (such as an attache case).

Table 11-1 defines the conditions under which the Apple IIc is designed to function properly.

*Table 11-1. Summary of Environmental Specifications*

Operating Temperature:	10° to 40° C (50° to 104° F)
Relative Humidity:	20% to 95%
Line Voltage:	105 to 129 VAC (normal USA voltage range)

You should treat the Apple IIc with the same kind of care as any other electrical appliance. You should protect it from physical abuse, and be careful not to bump it against furniture when you move it around. Put it in an attache case or other

protective covering if you carry it outside. You should also protect the mechanical keyboard and the electrical connectors inside the case from spilled liquids.

In normal operation (with the handle locked in its down position), enough air flows through the openings in the case to keep the insides from getting too hot. If you manage to overheat your Apple IIc—for example, by blocking the upper or lower ventilation openings—the first symptom will be erratic operation, such as unexpectedly changed data. The memory devices in the Apple IIc are especially sensitive to heat.

Disks are another heat-sensitive element of the system. If the built-in drive becomes too hot, a disk within can warp or even melt.

---

## **11.2 Power Requirements**

The electrical power that the Apple IIc, and everything that draws power from it, is limited by the tolerances of its power supply and internal voltage converter. This section describes these limits for the USA external power supply. Appendix G describes them for models built for other countries. The internal voltage converter is the same on all models.

---

### ***11.2.1 The External Power Supply***

If you purchased your Apple IIc outside the USA, consult Appendix G for external power supply characteristics.

The external power supply operates on normal household AC power and provides DC power to the Apple IIc internal converter. The basic specifications of the external power supply are listed in Table 11-2. The Apple IIc external power supply's cord must be plugged into a three-wire 115-volt (nominal) outlet. The line voltage must be in the range given in Table 11-2.



### Warning

**Important Safety Instructions:** This product is equipped with a three-wire grounding-type plug—a plug having a third (grounding) pin. This plug will only fit into a grounding-type AC outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding-type plug.

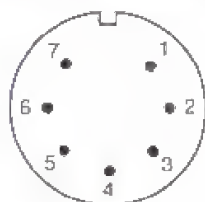
**Table 11-2.** Power Supply Specifications

Line Voltage:	105 to 129 VAC, 60 Hz
Maximum Input Power Consumption:	25 W
Supply Voltage:	+15 VDC (nominal)
Supply Current:	1.2 A (nominal)

### 11.2.2 The External Power Connector

The external power supply is attached to the internal converter by means of a 7-pin DIN connector. The connector pins are identified in Figure 11-1 and Table 11-3.

**Figure 11-1.** External Power Connector



Pin	Signal
1	Not connected
2,3	Signal ground
4	Shield ground
5,6	+15 VDC
7	Not connected

**Table 11-3. External Power Connector Signals**

Pin Number	Name	Description
1,7		Not connected
2,3	Ground	Common electrical ground
4	Chassis	Chassis ground
5,6	+15V	+15 volt DC input to converter

### **11.2.3 The Internal Converter**

The internal converter in the Apple IIc operates on from 9 to 20 volts DC as provided by the external power supply or its equivalent. The internal converter provides enough low-voltage electrical power for the built-in electronics plus an external disk drive attached via the 19-pin connector. The basic specifications of the internal converter are listed in Table 11-4. Listed amperages are those available in addition to the current drawn by the Apple IIc itself. Minus 5 volts is derived from the -12 volts provided by the voltage converter.

**Table 11-4. Internal Converter Specifications**

Input Voltage:	+9 to 20 VDC	
Maximum Power Consumption:	25 W	
Supply Voltages:	+5V	+5%
	+12V	+10%
	-12V	±10%
Maximum Supply Currents:	+5V:	1.5 A
	+12V:	0.6 A continuous 0.9 A intermittent 1.5 A surge (for < 100 ms)
	-12V:	100 mA
	(-5V:	50 mA)
Maximum Case Temperature:	60°C	(140°F)

The Apple IIc uses a switching-type internal voltage converter. It is small and lightweight, and it generates less heat than other types of voltage converters do.

The Apple IIc's voltage converter works by using the DC voltage input to power a variable-frequency oscillator. The oscillator drives a small transformer with several separate windings to produce the different voltages required. A circuit compares the voltage of the +5 volt supply with a reference voltage and feeds an error signal back to the oscillator circuit.

The oscillator circuit uses the error signal to control the duty cycle of its oscillation and keep the output voltages in their normal ranges.

The converter includes circuitry to protect itself and the other electronic parts of the Apple IIc by limiting all three output voltages whenever it detects one of the following malfunctions:

- any supply voltage short-circuited to ground
- any output voltage outside the normal range.

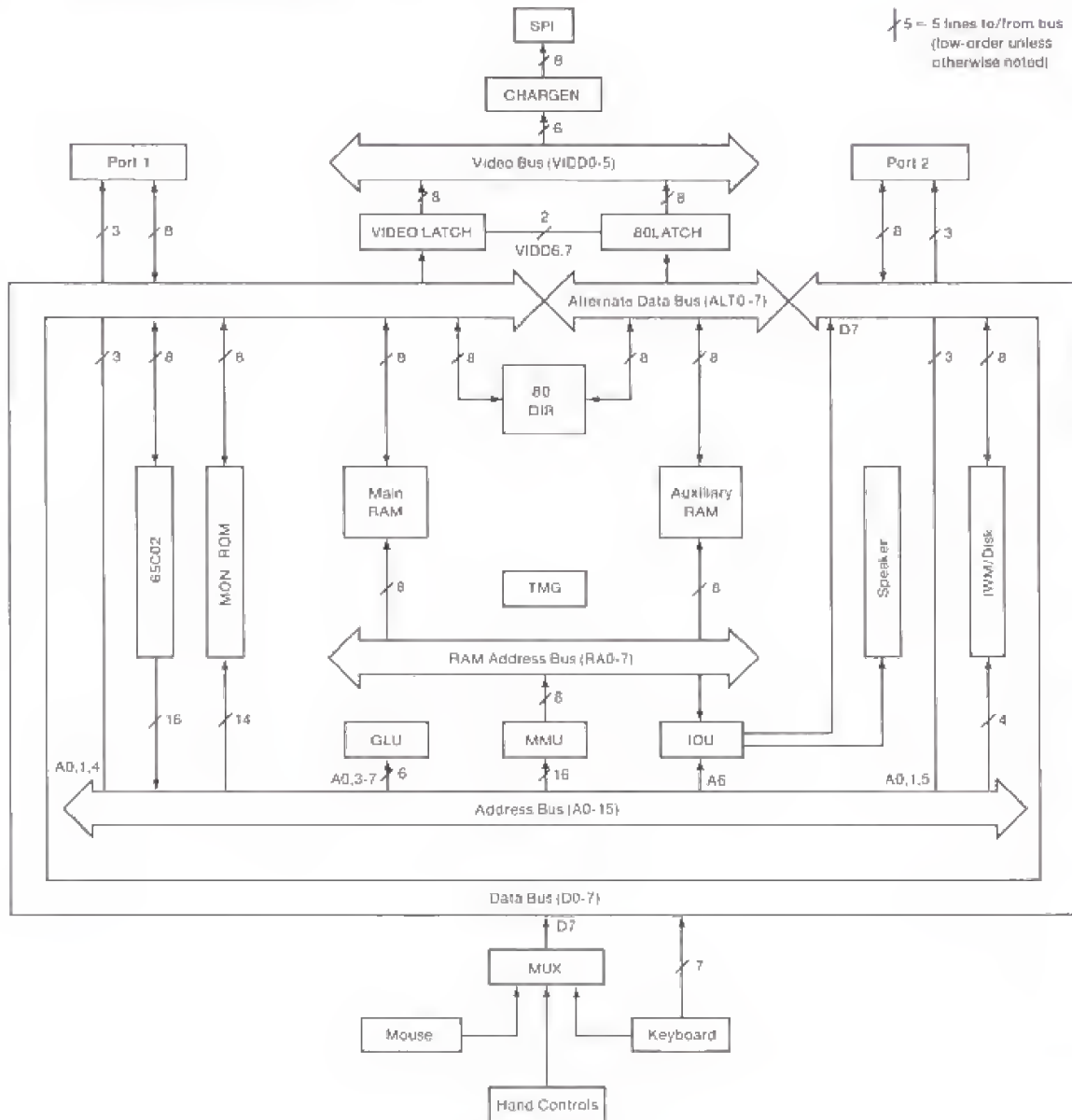
Any time one of these malfunctions occurs, the protection circuit varies the duty cycle of the oscillator, and all the output voltages drop to zero.

## 11.3 Apple IIc Overall Block Diagram

A full set of schematic diagrams of the Apple IIc appears in section 11.14.

Figure 11-2 is an overall block diagram of the Apple IIc. The following sections contain more detailed diagrams of the major parts of the machine.

Figure 11-2. Apple IIc Block Diagram





## 11.4 The CMOS 65C02 Microprocessor

The Apple IIc uses a CMOS 6502 (designated as 65C02) microprocessor as its central processing unit (CPU). The 65C02 in the Apple IIc runs at a clock rate of 1.023 MHz and performs up to 500,000 eight-bit operations per second.

**Note:** You should not use the clock rate as a criterion for comparing different types of microprocessors. The 65C02 has a simpler instruction cycle than most other microprocessors and it uses instruction pipelining for faster processing. The speed of the 65C02 with a 1 MHz clock is equivalent to many other types of microprocessors with clock rates up to 5 MHz.

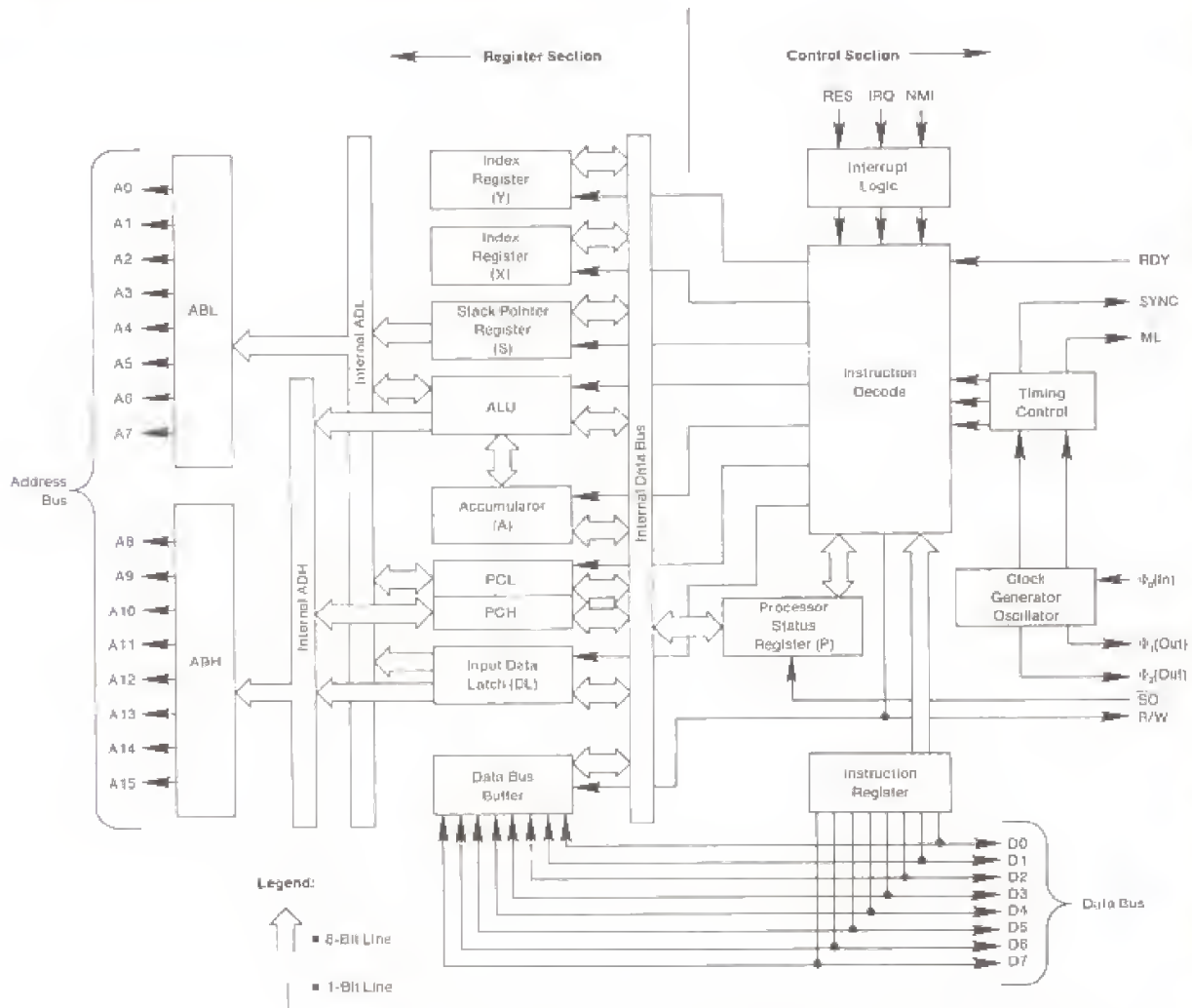
These instructions are described in Appendix A.

In addition to requiring lower power than earlier NMOS 6502 processors, the 65C02 in the Apple IIc provides the programmer with 27 new instructions. However, programs that use these additional instructions will not be backward compatible with other Apple II series computers that are not equipped with a CMOS 6502.

### 11.4.1 65C02 Block Diagram

Figure 11-3 is a block diagram of the 65C02 microprocessor. Table 11-5 contains the general specifications of this chip.

*Figure 11-3. 65C02 Block Diagram. Copyright 1982, NCR Corporation. Used by permission of NCR Corporation, Dayton, Ohio.*



The 65C02 has a sixteen-bit address bus, giving it an address space of 64K (two to the sixteenth power or 65536) bytes. The Apple IIc uses special techniques to address a total of more than 64K: for details, refer to Chapter 2.

**Table 11-5. 65C02 Microprocessor Specifications**

Type:	65C02
Register Complement:	8-bit Accumulator (A) 8-bit Index Registers (X,Y) 8-bit Stack Pointer (S) 8-bit Processor Status (P) 16-bit Program Counter (PC)
Data Bus:	Eight bits wide
Address Bus:	Sixteen bits wide
Address Range:	65,536 (64K)
Interrupts:	IRQ (maskable) NMI (non-maskable) BRK (programmed)
Operating Voltage:	1-5V ( $\pm 5\%$ )
Power Dissipation:	5 mW (at 1 MHz)

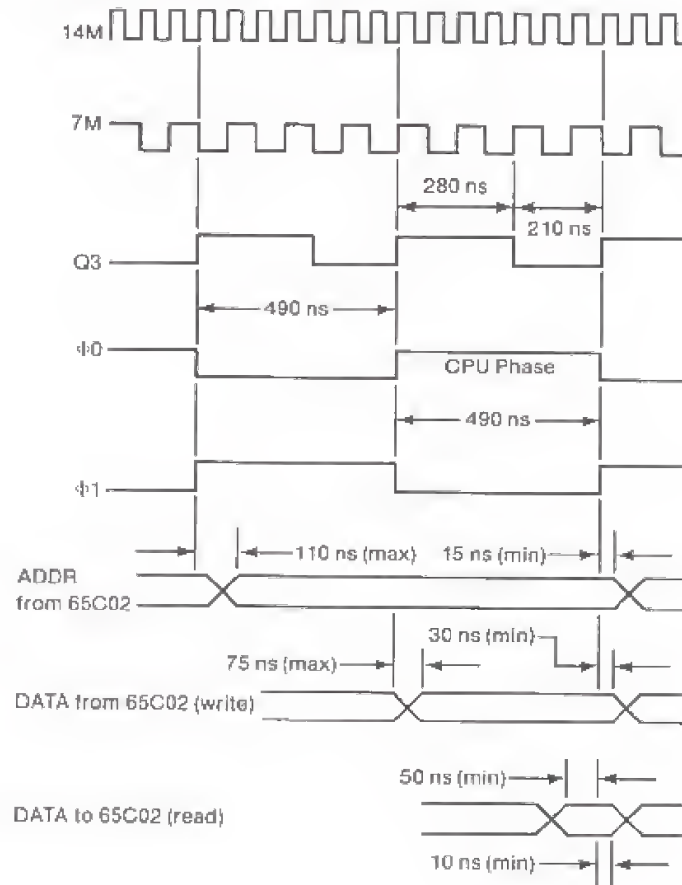
---

### **11.4.2 65C02 Timing**

The operation of the Apple IIc is controlled by a set of synchronous timing signals, sometimes called clock signals. The frequency of the oscillator that generates the master timing signal is 14.318 MHz. Circuitry in the Apple IIc uses this clock signal, called 14M, to produce all the other timing signals. These timing signals perform two major tasks: controlling the computing functions, and generating the video display. The timing signals directly involved with the operation of the 65C02 are described in this section. Other timing signals are described in sections 11.6.2, 11.9.3, and 11.9.4.

The main 65C02 timing signals are listed in Table 11-6, and their relationships are diagrammed in Figure 11-4. The 65C02 clock signals are  $\phi 1$  and  $\phi 0$ , complementary signals at a frequency of 1.0227 MHz. The Apple IIc signal named  $\phi 0$  is equivalent to the signal called  $\phi 2$  in Appendix A (it isn't identical—it's a tiny bit early).

Figure 11-4. 65C02 Timing Signals



**Table 11-6.** 65C02 Timing Signal Descriptions

Signal Name	Description
14M	Master oscillator, 14.318 MHz; also 80-column dot clock
VID7M	Intermediate timing signal and 40-column dot clock
Q3	Intermediate timing signal, 2.045 MHz with asymmetrical duty cycle
$\phi 0$	Phase 0 of 65C02 clock, 1.0227 MHz; complement of $\phi 1$
$\phi 1$	Phase 1 of 65C02 clock, 1.0227 MHz; complement of $\phi 0$

The operations of the 65C02 are related to the clock signals in a simple way: internal during  $\phi 1$ , external during  $\phi 0$ . The 65C02 puts an address on the address bus during  $\phi 1$ . This address is valid not later than 110 nanoseconds after  $\phi 1$  goes high and remains valid through all of  $\phi 0$ . The 65C02 reads or writes data during  $\phi 0$ . If the 65C02 is writing, the read/write signal is low during  $\phi 0$  and the 65C02 puts data on the data bus. The data is valid not later than 75 nanoseconds after  $\phi 0$  goes high. If the 65C02 is reading, the read/write signal remains high. Data on the data bus must be valid no later than 50 nanoseconds before the end of  $\phi 0$ .

## 11.5 The Custom Integrated Circuits

Most of the circuitry that controls memory and I/O addressing in the Apple IIc is in five custom integrated circuits

- the Memory Management Unit (MMU)
- the Input-Output Unit (IOU)
- the Timing Generator (TMG)
- the General Logic Unit (GLU)
- the Disk Controller Unit (IWM)

The soft switches used for controlling the various I/O and addressing modes of the Apple IIc are addressable flags inside the MMU, IOU, and GLU. The functions of the MMU and IOU are not as independent as their names suggest; working together, they generate all of the addressing signals. For example, the MMU generates the RAM address signals for the CPU, while the IOU generates similar RAM address signals for the video display and most I/O hardware addresses.

---

### 11.5.1 The Memory Management Unit (MMU)

The circuitry inside the MMU implements these soft switches, which are described in the following chapters:

- Page 2 display (PAGE2): Chapter 5
- High-resolution mode (HIRES): Chapter 5
- Store to 80-column display (80STORE): Chapter 5
- Select bank 2 (BANK2): Chapter 2
- Enable bank-switched RAM (ENLCRAM): Chapter 2
- Read auxiliary memory (RAMRD): Chapter 2
- Write auxiliary memory (RAMWRT): Chapter 2
- Auxiliary stack and zero page (ALTZP): Chapter 2
- Reset mouse Y interrupt (RSTYINT): Chapter 9
- Reset mouse X interrupt (RSTXINT): Chapter 9

These switches are available on MMU pin 21, which is connected to bit 7 on the data bus. Figure 11-5 shows the MMU pinouts; Table 11-7 describes the signals.

**Note:** A signal name followed by an asterisk is active low, that is, it is true when not asserted.

The 64K dynamic RAMs used in the Apple IIc use a multiplexed address, as described below in the section *Dynamic-RAM Timing* (in section 11.6.2). The MMU generates this multiplexed address for memory reading and writing by the 65C02 CPU.



**Figure 11-5. The MMU Pinouts**

GND	1	40	A1
A0	2	39	A2
$\phi 0$	3	38	A3
Q3	4	37	A4
PRAS*	5	36	A5
RA0	6	35	A6
RA1	7	34	A7
RA2	8	33	A8
RA3	9	32	A9
RA4	10	31	A10
RA5	11	30	A11
RA6	12	29	A12
RA7	13	28	A13
R/W*	14	27	A14
INH*	15	26	A15
C06X*	16	25	+5V
EN80*	17	24	SELIO*
KBD*	18	23	CASEN*
ROMEN2*	19	22	C07X*
ROMEN1*	20	21	MD7

**Table 11-7. The MMU Signal Descriptions**

Pin Number	Name	Description
1	GND	Power and signal common
2	A0	65C02 address input
3	$\phi 0$	Clock phase 0 input
4	Q3	Timing signal Input
5	PRAS*	Memory row-address strobe
6-13	RA0-RA7	Multiplexed address output
14	R/W*	65C02 read-write control signal
15	INH*	Inhibits main memory (tied to +5 V)
16	C06X*	Causes \$C06x outputs to go to 0 during $\phi 0$
17	EN80*	Enables auxiliary RAM
18	KBD*	Enables keyboard data bits 0-6
19	ROMEN2*	Enables ROM (tied to ROMEN1*)
20	ROMEN1*	Enables ROM (tied to ROMEN2*)
21	MD7	State of MMU flags on data bus bit 7
22	C07X	Causes \$C07x outputs to go to 0 during $\phi 0$
23	CASEN*	Enables main RAM
24	SELIO*	Goes to 0 during $\phi 0$ for any access to \$C0 page except \$C08x, Bx, Cx or Fx
25	+5V	Power
26-40	A15-A1	65C02 address Input

### 11.5.2 The Input/Output Unit (IOU)

The circuitry inside the Input/Output Unit (IOU) implements the following soft switches, all described in Chapters 2 and 3:

- Page 2 display (PAGE2)
- High-resolution mode (HIRES)
- Text mode (TEXT)
- Mixed mode (MIXED)
- 80-column display (80COL)
- Character-set select (ALTCHAR)
- Any-key-down (AKD)
- Mouse movement (X0, Y0)
- Vertical blanking interrupt (VBLINT)

These switches are available on IOU pin 9, which is connected to bit 7 on the data bus. Figure 11-6 shows the MMU pinouts; Table 11-8 describes the signals.

The 64K dynamic RAMs used in the Apple IIc require a multiplexed address, as described in the section *Dynamic-RAM Timing* (in section 11.6.2). The IOU generates this multiplexed address for the data transfers required for display and memory refresh during clock phase 1. The way this address is generated is described in section 11.9.1.

Figure 11-6. The IOU Pinouts

GND	1	40	H0
GR	2	39	SYNC*
SEGA	3	38	WINDW*
SEGB	4	37	CLRGAT*
VC	5	36	RA10*
80COL*	6	35	RA9*
CASSQ	7	34	VIDD6
SPKR	8	33	VIDD7
MD7	9	32	KSTRE
YMOVE	10	31	AKD
(N.C.)	11	30	IOUSELIO*
(N.C.)	12	29	A6
PDL0/XMOVE	13	28	+5V
R/W*	14	27	Q3
RESET*	15	26	Q0
IRQ*	16	25	PRAS*
RA0	17	24	RA7
RA1	18	23	RA6
RA2	19	22	RA5
RA3	20	21	RA4

Table 11-8. The IOU Signal Descriptions

Pin Number	Name	Description
1	GND	Power and signal common
2	GR	Graphics mode enable
3	SEGA	In text mode, works with VC (see pin 5) and SEGB to determine character row address
4	SEGB	In text mode, works with VC (see pin 5) and SEGA; in graphics mode, selects high-resolution when low, low-resolution when high
5	VC	Display vertical counter bit; in text mode, SEGA, SEGB and VC determine which of the eight rows of a character's dot pattern to display; in low-resolution, selects upper or lower block defined by a byte.

*Table 11-8—Continued. The IOU Signal Descriptions*

Pin Number	Name	Description
6	B0COL*	80-column video enable
7	CASSO	Reserved
8	SPKR	Speaker output signal
9	MD7	Internal IOU flags for data bus (bit 7)
10	YMOVE	Detects mouse movement along Y axis
11	N.C.	Not used
12	N.C.	Not used
13	PDL0/XMOVE	Detects mouse movement along X axis
14	R/W*	65C02 read-write control signal
15	RESET*	Power on and reset output
16	IRQ*	Maskable interrupt line to 65C02
17-24	RA0-RA7	Video refresh multiplexed RAM address (phase 1)
25	PRAS*	Row-address strobe (phase 0)
26	$\phi 0$	Master clock phase 0
27	Q3	Intermediate timing signal
28	+5V	Power
29	A6	Address bit 6 from 65C02
30	IOUSELIO*	Derived from the SELIO* output for MMU pin 24
31	AKD	Any-key-down signal
32	KSTRB	Keyboard strobe signal
33,34	VIDD7,VIDD6	Video display data bits
35,36	RA9*,RA10*	Video display control bits
37	CLRGAT*	Color-burst gate (enable)
38	WNDW*	Display blanking signal
39	SYNC*	Display synchronization signal
40	H0	Display horizontal timing signal (low bit of character counter)

### 11.5.3 The Timing Generator (TMG)

A custom timing generator chip (TMG) generates several timing and control signals in the Apple IIc. The TMG pinouts are shown in Figure 11-7; the signals are listed in Table 11-9.

**Figure 11-7.** The TMG Pinouts

14M	1	20	+5V
7M	2	19	PRAS*
CREF	3	18	(N.C.)
H0	4	17	PCAS*
VIDD7	5	16	Q3
SEGB	6	15	$\phi 0$
TEST	7	14	$\phi 1$
CASEN*	8	13	VID7M
80COL*	9	12	LDPS*
GND	10	11	TMGEN*

**Table 11-9.** The TMG Signal Descriptions

Pin Number	Name	Description
1	14M	14.318 MHz master timing signal input
2	7M	7.159 MHz timing signal
3	CREF	3.5795 MHz color reference timing signal
4	H0	Horizontal video timing signal
5	VIDD7	Video data bit 7
6	SEGB	Video timing signal
7	TEXT	Video display text-modes enable
8	CASEN*	RAM enable (CAS enable)
9	80COL*	Enable 80-column display mode
10	GND	Power and signal common
11	TMGEN*	Enable master timing
12	LDPS*	Video shift-register load enable
13	VID7M	Video dot clock enable, 7 MHz or continuous 0
14	$\phi 1$	Phase 1 system clock
15	$\phi 0$	Phase 0 system clock
16	Q3	Intermediate timing and strobe signal
17	PCAS*	RAM column-address strobe
18	N.C.	Reserved for testing
19	PRAS*	RAM row-address strobe
20	+5V	Power

### 11.5.4 The General Logic Unit (GLU)

The General Logic Unit is a single-chip version of the miscellaneous logic required for the system. It provides all RAM read/write timing, double-high-resolution enable/disable, soft switch status registers and write command registers. It also provides IOU control for mouse interrupts and double-high-resolution soft-switches. Its pin assignments are shown in Figure 11-8 and its signals are listed in Table 11-10.

Figure 11-8. The GLU Pinouts

14M	1	24	+5V
A0	2	23	SER*
A3	3	22	(N.C.)
A4	4	21	DISK*
A5	5	20	7M
A6	6	19	CREF
A7	7	18	(N.C.)
I/O	8	17	(N.C.)
SELIO*	9	16	TEXT
GR	10	15	R/W*
RESET*	11	14	D7
GND	12	13	GLUEN*

Table 11-10. The GLU Signal Descriptions

Pin Number	Name	Description
1	14M	Master clock (14.318 MHz)
2,3-7	A0,A3-A7	Address lines to select least significant byte of addresses on C0 page
8	PH0	Phase 0 of 1.0227 MHz processor sync clock
9	SELIO*	Device select for selecting most significant byte of the address
10	GR	Graphics mode select line
11	RESET*	Master reset for system; resets GLU
12	GND	Ground reference and negative supply
13	GLUEN*	Enables GLU
14	MD7	Indicates status of MMU flags on data bus bit 7
15	R/W*	Read/write qualifier input from processor
16	TEXT	Signal used to generate video timing in double-high-resolution or not-graphics
17,18	N.C.	Not used
19	CREF	Color reference signal
20	7M	7 MHz clock output
21	DISK*	Disk controller device select output
22	IOUHOLE	Controls IOUSELIO
23	SER*	Serial controller device select output
24	Vcc	+5 volt supply

For further information on group code recording, refer to section 11.10.

### 11.5.5 The Disk Controller Unit (IWM)

The IWM is an integrated GCR (group code recording) disk drive controller in its state right after reset. In addition, it has a status register, mode register, and multiple operating modes. It provides both synchronous and asynchronous modes, and a fast mode with a data rate twice that of normal disk I/O speeds. Figure 11-9 shows the IWM pin assignments; Table 11-11 describes the IWM signals.

Figure 11-9. The IWM Pinouts

SEEKPH0	1	28	SEEKPH1
SEEKPH2	2	27	SEEKPH3
A0	3	26	+5V
A1	4	25	Q3
A2	5	24	7M
A3	6	23	RESET*
DISK*	7	22	RDDATA
WRDATA	8	21	WRPROT
WRREQ*	9	20	DR1*
D0	10	19	DR2*
D1	11	18	D7
D2	12	17	D6
D3	13	16	D5
GND	14	15	D4

Table 11-11. The IWM Signal Descriptions

Pin Number	Name	Description
1	SEEKPH0	Stepper motor control phase 0. One of four programmable disk drive motor phase outputs.
2	SEEKPH2	Stepper motor control phase 2
3	A0	The data input to the state bit selected by A1 to A3
4-6	A1-A3	These three inputs select one of the eight bits in the state register to be updated.
7	DISK*	Device enable. The falling edge of DISK* latches information on A1-A3. The rising edge of either Q3 or DISK* qualifies write register data.
8	WRDATA	The serial data output. Each 1-bit causes a transition on this output.
9	WRREQ*	This signal is a programmable buffered output line.
10-13	D0-D3	D0-D7 make up the bidirectional data bus.
14	GND	Ground reference and negative supply
15-18	D4-D7	The remaining bits of the bidirectional data bus
19	DR2*	Drive 2 select
20	DR1*	Drive 1 select
21	WRPROT	Write-protect input; this can be polled via bit 7 of the status register.



*Table 11-11—Continued. The IWM Signal Descriptions*

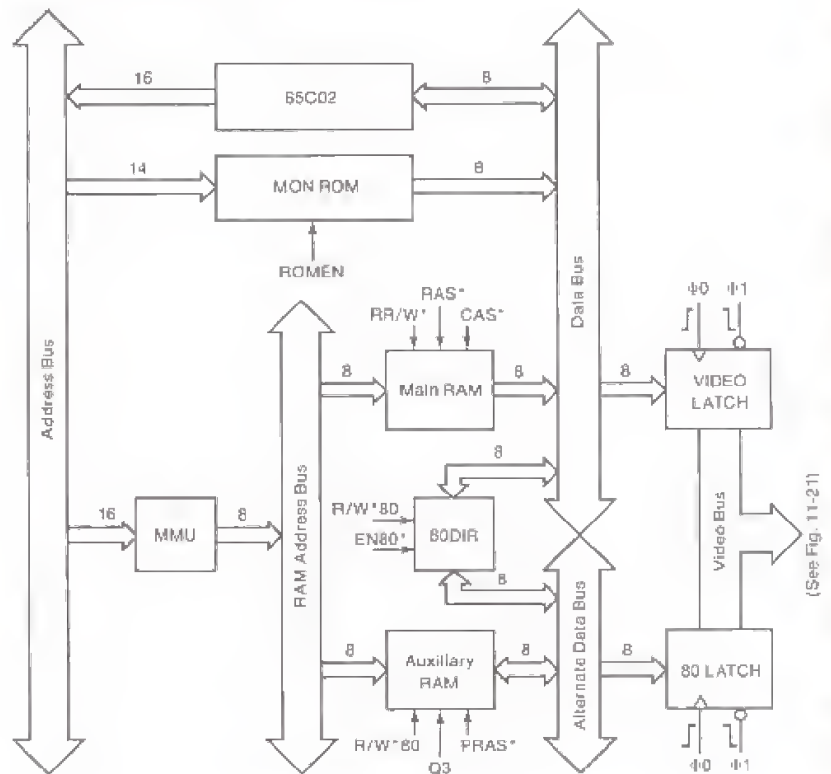
Pin Number	Name	Description
22	RDDATA	Serial data input line. The IWM synchronizes the falling transition of each pulse.
23	RESET*	IWM reset: places all IWM outputs in their inactive state and sets all state and mode register bits to zero.
24	7M	7 MHz clock input
25	Q3	A 2.0 MHz clock input used to qualify the timing of the serial data being written or read.
26	Vcc	The +5 volt supply
27	SEEKPH3	Stepper motor control phase 3
28	SEEKPH1	Stepper motor control phase 1

## 11.6 Memory Addressing

The 65C02 microprocessor can address 65,536 locations. The Apple IIc uses this entire address space, and then some: some areas in memory are used for more than one function. The following sections describe the memory devices used in the Apple IIc and the way they are addressed. Input and output also use portions of the memory address space; refer to Chapter 2 for information.

Figure 11-10 illustrates the overall memory bus organization and memory selection signals.

*Figure 11-10. Memory Bus Organization*



**Note:** Some Apple IIc's have ROMs with 27xx designations, some have 23xx. They are functionally equivalent.

### 11.6.1 ROM Addressing

In the Apple IIc the following programs are permanently stored in a type 23128 16K by 8-bit ROM (Figure 11-11).

- Applesoft editor and interpreter
- Monitor
- Enhanced video firmware.

**Figure 11-11. The 23128 ROM Pinouts**

+5V	1	28	+5V
A12	2	27	(N.C.)
A7	3	26	A13
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	22	OE*
A2	8	21	A10
A1	9	20	CE*
A0	10	19	D7
D0	11	18	D6
D1	12	17	D5
D2	13	16	D4
GND	14	15	D3

**Figure 11-12. The 2316 ROM Pinouts**

KA7	1	24	+5V
KA6	2	23	KA8
KA5	3	22	CAPS
KA4	4	21	+5V
KA3	5	20	KBD*
KA2	6	19	LANGSW
KA1	7	18	GND
KA0	8	17	(N.C.)
D0	9	16	D6
D1	10	15	D5
D2	11	14	D4
GND	12	13	D3

**Figure 11-13. The 2364 ROM Pinouts**

+5V	1	28	+5V
A12	2	27	+5V
A7	3	26	+5V
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	22	GND
A2	8	21	A10
A1	9	20	WNDW*
A0	10	19	O7
O0	11	18	O6
O1	12	17	O5
O2	13	16	O4
GND	14	15	O3

The ROM is enabled by two signals called ROMEN1 and ROMEN2. (In the Apple IIc, ROMEN1 and ROMEN2 are electrically connected.) The segment of the ROM enabled by ROMEN1 occupies the memory address space from \$C100 to \$DFFF. The address space from \$C300 to \$C3FF and much of \$C800 to \$CFFF contains the enhanced video firmware.

These ROM address allocations are approximately true (some space sharing takes place):

- ROM addresses \$C000 to \$C0FF are never available.
- ROM addresses \$C100 and \$C200 are entry points to firmware for serial ports 1 and 2, respectively.
- ROM address \$C400 is entry point to mouse interface support.
- ROM addresses \$C500 to \$C5FF are reserved.
- ROM address \$C600 is entry point to firmware for the built-in and external disk drives. The built-in drive is considered slot 6 drive 1 or its equivalent. The external drive is considered slot 6 drive 2.
- ROM addresses starting at \$C700 support (from the Monitor) the external drive as if it were slot 7 drive 1, for external-drive startup only.

Addresses \$D000 to \$F7FF contain the Applesoft BASIC interpreter; addresses \$F800 through \$FFFF contain the Monitor firmware.

The other ROMs in the Apple IIc are a type 2316 ROM (Figure 11-12) used for the keyboard character decoder, and a type 2364 ROM (Figure 11-13) used for character sets for the video display. This 2364 ROM is rather large because it includes a section of straight-through bit-mapping for the graphics modes. This way, graphics display video can pass through the same circuits as text without additional switching circuitry.

---

### 11.6.2 RAM Addressing

The RAM (programmable) memory in the Apple IIc is used both for program and data storage and for the video display. The areas in RAM that are used for the display are accessed both by the 65C02 microprocessor and by the video display circuits. In some computers, this dual access results in addressing conflicts (cycle stealing) that can cause temporary dropouts in the video display. This problem does not occur in the Apple IIc, thanks to the way the microprocessor and the video circuits share the memory.

The memory circuits in the Apple IIc take advantage of the two-phase system clock described in section 11.4.2 to interleave the microprocessor memory accesses and the display memory accesses so that they never interfere with each other. The microprocessor reads or writes to RAM only during  $\phi 0$ , and the display circuits read data only during  $\phi 1$ .

---

### Dynamic-RAM Refreshment

The image on a video display is not permanent; it fades rapidly and must be refreshed periodically. To refresh the video display, the Apple IIc reads the data in the active display page and sends it to the display. To prevent visible flicker in the display, and to conform to standard practice for broadcast video, the Apple IIc refreshes the display sixty times per second.

The dynamic RAM devices used in the Apple IIc also need a kind of refreshment, because the data is stored in the form of electric charges which diminish with time and must be replenished every so often. The Apple IIc is designed so that refreshing the display also refreshes the dynamic RAMs. The next few paragraphs explain how this is done.

The job of refreshing the dynamic RAM devices is minimized by the structure of the devices themselves. The individual data cells in each RAM device are arranged in a rectangular array of rows and columns. When the device is addressed, the part of the address that specifies a row is presented first, followed by the address of the column. Splitting information into parts that follow each other in time is called **multiplexing**. Since only half of the address is needed at one time, multiplexing the address reduces the number of pins needed for connecting the RAMs (Figure 11-14).

**Figure 11-14.** The 64K RAM Pinouts

+5V	1	16	GND
MDx	2	15	CAS*
R/W*	3	14	MDx
RAS*	4	13	RA1
RA7	5	12	RA4
RA5	6	11	RA3
RA6	7	10	RA2
+5V	8	9	RA0

Different manufacturers' 64K RAMs have cell arrays of either 128 rows by 512 columns or 256 rows by 256 columns. Only the row portion of the address is used in refreshing the RAMs.

Now consider how the display is refreshed. As described in section 11.9.1, the display circuitry generates a sequence of 8,192 memory addresses in high-resolution mode; in text and low-resolution modes, this sequence is the 1,024 display-page addresses repeated eight times. The display address cycles through this sequence 60 times a second, or once every 17 milliseconds. The way the low-order address lines are assigned to the RAMs, the row address cycles through all 256 possible values once every two milliseconds (see Table 11-12). This more than satisfies the refresh requirements of the dynamic RAMs.

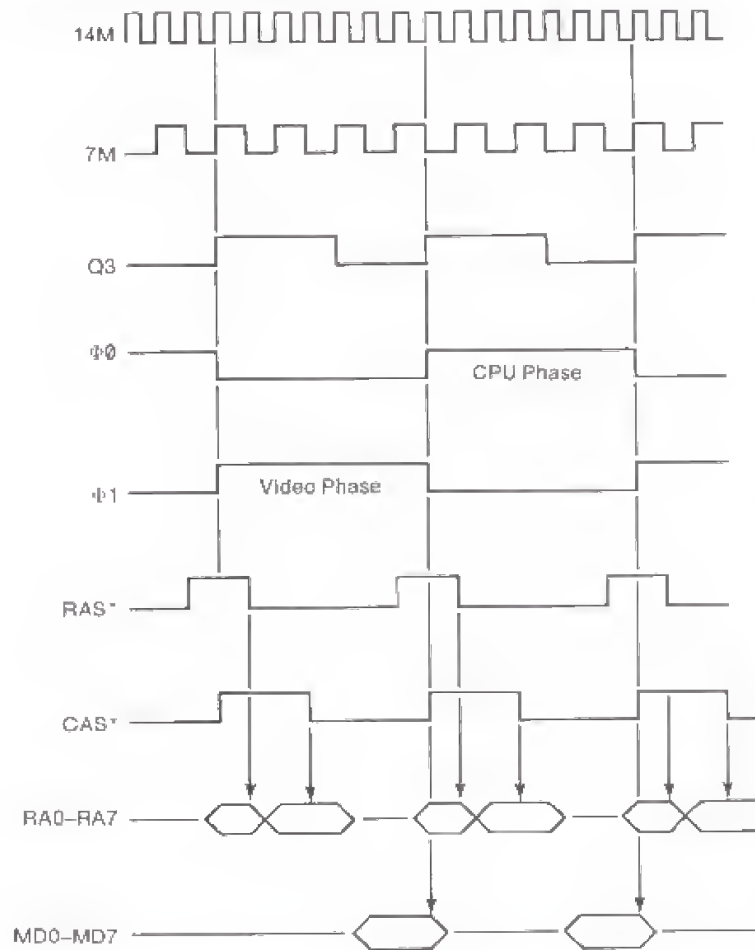
**Table 11-12.** RAM Address Multiplexing

Mux'd Address	Row Address	Column Address
RA0	A0	A9
RA1	A1	A6
RA2	A2	A10
RA3	A3	A11
RA4	A4	A12
RA5	A5	A13
RA6	A7	A14
RA7	A8	A15

### ***Dynamic-RAM Timing***

The Apple IIc's microprocessor clock runs at a speed of 1.023 MHz, but the interleaving of CPU and display cycles means that the RAM is being accessed at a 2 MHz rate, or a cycle time of just under 500 nanoseconds. Data for the CPU is strobed by the falling edge of  $\phi 0$ , and display data is strobed by the falling edge of  $\phi 1$ , as shown in Figure 11-15.

Figure 11-16. RAM Timing Signals



The RAM timing looks complicated because the RAM address is multiplexed, as described in the previous section. The MMU takes care of multiplexing the address for the CPU cycle, and the IOU performs the same function for the display cycle. The multiplexed address is sent to the RAM ICs over the lines labeled RA0-RA7 (Table 11-13). Along with the other timing signals, the TMG generates two signals that control the RAM addressing: row-address strobe (RAS) and column-address strobe (CAS).



*Table 11-13. RAM Timing Signals*

Signal Name	Description
$\phi 0$	Clock phase 0 (CPU phase)
$\phi 1$	Clock phase 1 (display phase)
RAS	Row-address strobe
CAS	Column-address strobe
Q3	Alternate RAM/Column-address strobe
RA0-RA7	Multiplexed address bus
MD0-MD7	Internal data bus

## 11.7 The Keyboard

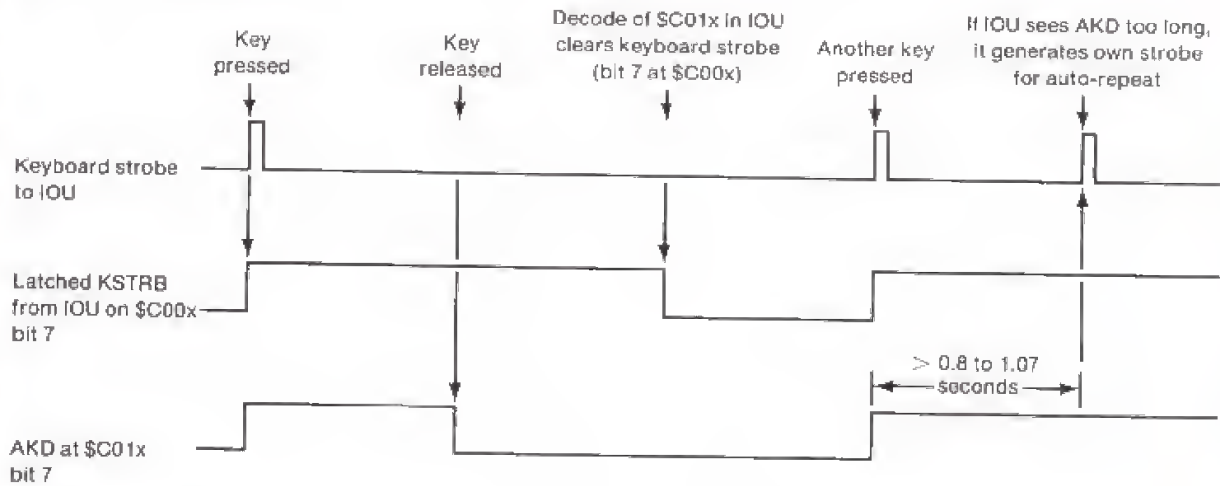
The Apple IIc's keyboard is a matrix of keyswitches connected to an AY-3600-type keyboard decoder via a ribbon cable and a 26-pin connector (Figure 11-16). The AY-3600 scans the array of keys over and over to detect any keys pressed. The scanning rate is set by the external resistor-capacitor network made up of C46 and R6. The debounce time is also set externally, by C45.

The AY-3600's outputs include five bits of key code plus separate lines for CONTROL, SHIFT, any-key-down, and keyboard strobe. The any-key-down and keyboard-strobe lines are connected to the IOU, which addresses them as soft switches. The key-code line, along with CONTROL and SHIFT, are inputs to a separate 2316 ROM. The ROM translates them to the character codes that are enabled onto the data bus by signals named KBD\* and ENKBD\*. The KBD\* signal is enabled by the MMU whenever a program reads location \$C000, as described in Chapter 2.



Figure 11-17 illustrates the events that occur when a key is pressed, when the keypress is detected by a program, and when a key is pressed and held for more than about a second.

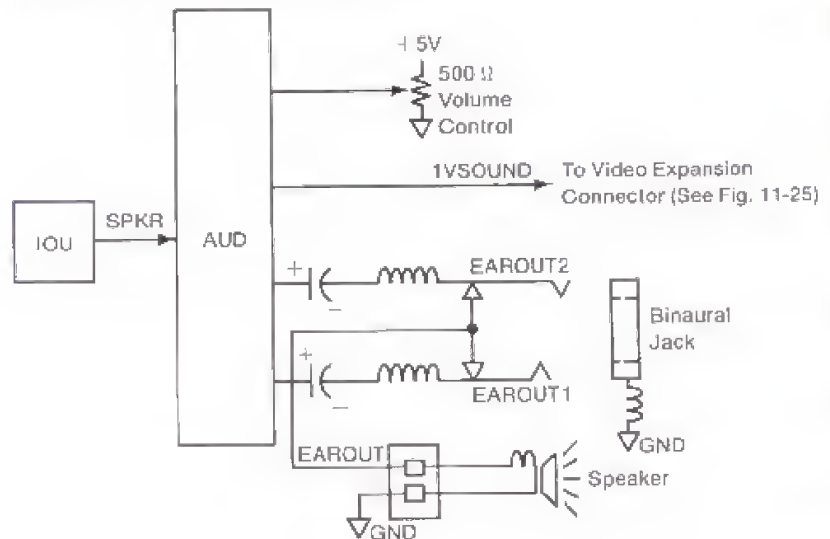
Figure 11-17. Keyboard Signals



## 11.8 The Speaker

The Apple IIc's built-in loudspeaker is controlled by a single bit of output from the Input/Output Unit (IOU), amplified by a hybrid circuit (Figure 11-18).

Figure 11-18. Speaker Circuit Diagram



AUD is an audio-amplifier hybrid circuit.

### 11.8.1 Volume Control

There is a 500-ohm variable resistor feeding anywhere from 0 to 5 volts to pin 5 of AUD to control the speaker volume. This potentiometer controls the volume of both the built-in speaker and whatever is plugged into the output jack.

### 11.8.2 Output Jack

Next to the volume control, along the lower-left side of the Apple IIc case, there is a 3.5 mm stereo miniphone jack. Although speaker output is monaural, the jack accommodates stereo headphone plugs (as well as monaural, of course), providing sound to both channels. Inserting a headphone plug disconnects the internal Apple IIc speaker.

---

## 11.9 The Video Display

The Apple IIc produces a video signal that creates a display on a standard video monitor or, if you add an RF modulator, on a black-and-white or color television set. The video signal is a composite made up of the data that is being displayed plus the horizontal and vertical synchronization signals that the video monitor uses to arrange the lines of display data on the screen.

**Note:** Apple IIc computers manufactured for sale in the USA generate a video signal that is compatible with the standards set by the NTSC (National Television Standards Committee). Apple IIcs used in European countries require an external adapter to provide video that is compatible with the standard used there, which is called PAL (for phase alternating lines). This manual describes only the NTSC version of the video circuits.

The display portion of the video signal is a time-varying voltage generated from a stream of data bits, where a 1 corresponds to a voltage that generates a bright dot, and a 0 to a dark dot. The display bit stream is generated in bursts that correspond to the horizontal lines of dots on the video screen. The signal named WNDW\* is low during these bursts.

During the time intervals between bursts of data, nothing is displayed on the screen. During these intervals, called the **blanking intervals**, the display is blank and the WNDW\* signal is high. The synchronization signals, called **sync** for short, are produced by making the signal named SYNC\* low during portions of the blanking intervals. The sync pulses are at a voltage equivalent to blacker-than-black video and don't show on the screen.

---

### 11.9.1 The Video Counters

The address and timing signals that control the generation of the video display are all derived from a chain of counters inside the IOU. Only a few of these counter signals are accessible from outside the IOU, but they are all important in understanding the operation of the display generation process, particularly the display memory addressing described in the next section.

The horizontal counter is made up of seven stages: H0, H1, H2, H3, H4, H5, and HPE\*. The input to the horizontal counter is the 1 MHz signal that controls the reading of data being displayed. The complete cycle of the horizontal counter consists of 65 states. The six bits H0 through H5 count normally from 0 to 64, then start over at 0. Whenever this happens, HPE\* forces another count with H0 through H5 held at zero, thus extending the total count to 65.

The IOU uses the forty horizontal count values from 25 through 64 in generating the low-order part of the display data address, as described in section 11.9.3. The IOU uses the count values from 0 to 24 to generate the horizontal blanking, the horizontal sync pulse, and the color-burst gate.

When the horizontal count gets to 65, it signals the end of a line by triggering the vertical counter. The vertical counter has nine stages: VA, VB, VC, V0, V1, V2, V3, V4, and V5. When the vertical count reaches 262, the IOU resets it and starts counting again from zero. Only the first 192 scanning lines are actually displayed; the IOU uses the vertical counts from 192 to 262 to generate the vertical blanking and sync pulse. Nothing is displayed during the vertical blanking interval. (The vertical line count is 262 rather than the standard 262.5 because, unlike normal television, the Apple IIc's video display is not interlaced.)

---

### ***11.9.2 Display Memory Addressing***

As described in section 5.7, data bytes are not stored in memory in the same sequence in which they appear on the display. You can get an idea of the way the display data is stored by using the Monitor to set the display to graphics mode, then storing data starting at the beginning of the display page at hexadecimal \$400 and watching the effect on the display. If you do this, you should use the graphics display instead of text to avoid confusion: the text display is also used for Monitor input and output.

If you want your program to display data by storing it directly into the display memory, you must first transform the display coordinates into the appropriate memory addresses, as shown in Chapter 2. The descriptions that follow will help you understand how this address transformation is done and why it is necessary.



The address transformation that folds three rows of forty display bytes into 128 contiguous memory locations is the same for all display modes, so it is described first. The differences among the different display modes are described in section 11.9.4.

---

### 11.9.3 Display Address Mapping

Consider the simplest display on the Apple IIc, the 40-column text mode. To address forty columns requires six bits, and to address twenty-four rows requires another five bits, for a total of eleven address bits. Addressing the display this way would involve 2048 (two to the eleventh power) bytes of memory to display a mere 960 characters. The 80-column text mode would require 4096 bytes to display 1920 characters. The leftover chunks of memory that were not displayed could be used for storing other data, but not easily, because they would not be contiguous.

Instead of using the horizontal and vertical counts to address memory directly, the circuitry inside the IOU transforms them into the new address signals described below. The transformed display address must meet the following criteria:

- Map the 960 bytes of 40-column text into only 1024 bytes
- Scan the low-order address to refresh the dynamic RAMs
- Continue to refresh the RAMs during video blanking.

The requirements for RAM refreshing are discussed in section 11.6.2.

The transformation involves only horizontal counts H3, H4, and H5, and vertical counts V3 and V4. Vertical count bits VA, VB, and VC address the lines making up the characters, and are not involved in the address transformation. The remaining low-order count bits, H0, H1, H2, V0, V1, and V2 are used directly, and are not involved in the transformation.

The IOU performs an addition that reduces the five significant count bits to four new signals called S0, S1, S2, and S3, where S stands for **sum**. Figure 11-19 is a diagram showing the addition in binary form, with V3 appearing as the carry in and H5 appearing as its complement H5\*. A constant value of one appears as the low-order bit of the addend. The carry bit generated with the sum is not used.

**Figure 11-19. Display Address Transformation**

			V3	Carry in
H5*	V3	H4	H3	Augend
V4	H5*	V4	1	Addend
S3	S2	S1	S0	Sum

If this transformation seems terribly obscure, try it with actual values. For example, for the upper-left corner of the display, the vertical count is zero and the horizontal count is 24: H0, H1, H2, and H5 are zeroes, and H3 and H4 are ones. The value of the sum is zero, so the memory location for the first character on the display is the first location in the display page, as you might expect.

Horizontal bits H0, H1, and H2 and sum bits S0, S1, and S2 make up the transformed horizontal address (A0 through A6 in Table 11-14). As the horizontal count increases from 24 to 63, the value of the sum (S3 S2 S1 S0) increases from zero to four and the transformed address goes from 0 to 39, relative to the beginning of the display page.

The low-order three bits of the vertical row counter are V0, V1, and V2. These bits control address bits A7, A8, and A9, as shown in Table 11-14, so that rows 0 through 7 start on 128-byte boundaries. When the vertical row counter reaches 8, V0, V1, and V2 are zero again, and V3 changes to one. If you do the addition in Figure 11-19 with H equal to 24 (the horizontal count for the first column displayed) and V equal to 8, the sum is 5 and the horizontal address is 40: the first character in row 8 is stored in the memory location 40 bytes from the beginning of the display page.

**Figure 11-20.** 40-Column Text Display Memory. Memory locations marked with a double asterisk (\*\*) are screen holes, described in section 2.5.1.

	128 Bytes			
	40 Bytes	40 Bytes	40 Bytes	8 Bytes
\$400	Row 0	Row 8	Row 16	**
\$480	Row 1	Row 9	Row 17	**
\$500	Row 2	Row 10	Row 18	**
\$580	Row 3	Row 11	Row 19	**
\$600	Row 4	Row 12	Row 20	**
\$680	Row 5	Row 13	Row 21	**
\$700	Row 6	Row 14	Row 22	**
\$780	Row 7	Row 15	Row 23	**

Figure 11-20 shows how groups of three forty-character rows are stored in blocks of 120 contiguous bytes starting on 128-byte address boundaries. This diagram is another way of describing the display mapping shown in Figure 5-5. Notice that the three rows in each block of 120 bytes are not adjacent on the display.

**Table 11-14. Display Memory Addressing.** \*\*For these address bits, see text and Table 11-15.

Memory Address Bit	Display Address Bit
A0	H0
A1	H1
A2	H2
A3	S0
A4	S1
A5	S2
A6	S3
A7	V0
A8	V1
A9	V2
A10	**
A11	**
A12	**
A13	**
A14	**
A15	GND

Address bits marked with double asterisks (\*\*) are different for different modes: see Table 11-15 and section 11.9.4.

Table 11-14 shows how the signals from the video counters are assigned to the address lines. H0, H1, and H2 are horizontal-count bits, and V0, V1, and V2 are vertical-count bits. S0, S1, S2 and S3 are the folded address bits described above.

**Table 11-15. Memory Address Bits for Display Modes.** (. means logical AND; ' means logical NOT.)

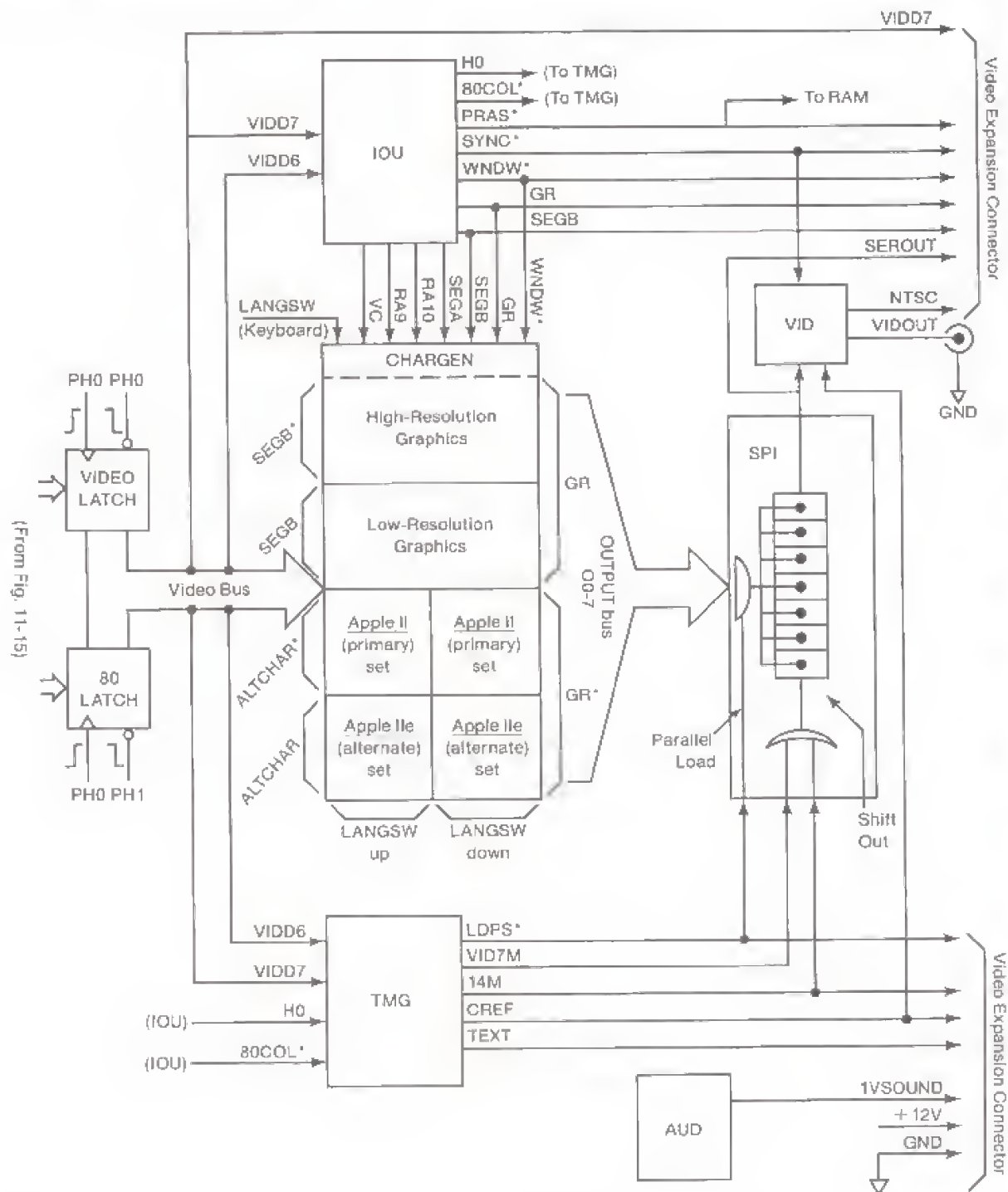
Address Bit	Display Modes	
	Text and Low-Resolution	High-Resolution and Double-High-Resolution
A10	80STORE + PAGE2'	VA
A11	80STORE'.PAGE2	VB
A12	0	VC
A13	0	80STORE + PAGE2'
A14	0	80STORE'.PAGE2

---

#### ***11.9.4 Video Display Modes***

The different display modes all use the address-mapping scheme described in section 11.9.3, but they use different-sized memory areas in different locations. Section 11.9.4 describes the addressing schemes and the methods of generating the actual video signals for the different display modes. Figure 11-21 illustrates the video display circuits discussed in this section.

Figure 11-21. Video Display Circuits





---

### *Text Displays*

The text and low-resolution graphics pages begin at memory locations \$400 and \$800. Table 11-15 shows how the display-mode signals control the address bits to produce these addresses. Address bits A10 and A11 are controlled by the settings of PAGE2 and 80STORE, the display-page and 80-column-video soft switches. Address bits A12, A13, and A14 are set to zero. Notice that 80STORE active inhibits PAGE2: there is only one display page in 80-column mode.

The low-order six bits of each data byte reach the character generator directly, via the video data bus VID0-VID5. The two high-order bits are modified by the IOU to select between the primary and alternate character sets and are sent to the character generator on lines RA9 and RA10.

The data for each row of characters are read eight times, once for each of the eight lines of dots making up the row of characters. The data bits are sent to the character generator along with VA, VB, and VC, the low-order bits from the vertical counter. For each character being displayed, the character generator puts out one of eight stored bit patterns selected by the three-bit number made up of VA, VB, and VC.

The bit patterns from the character generator are loaded into the 74166 parallel-to-serial shift register and output as a serial bit stream that goes to the video output circuit (Figure 11-21). The shift register is controlled by signals named LDPS\* (for load parallel-to-serial shifter) and VID7M (for video 7 Mhz). In 40-column mode, LDPS\* strobes the output of the character generator into the shift register once each microsecond, and VID7M shifts the bits out at 7 MHz (Figure 11-22).

The addressing for the 80-column display is exactly the same as for the 40-column display: the 40 columns of display memory in auxiliary memory are addressed in parallel with the 40 columns in main memory. The data from these two memories reach the video data bus (lines VID0-VID7) via separate 74LS374 three-state buffers. These buffers are loaded simultaneously (at the rising edge of  $\phi 0$ ), but their outputs are sent to the character generator alternately by the falling edge of  $\phi 0$  and  $\phi 1$ . In 80-column mode, LDPS\* loads data from the character generator into the shift register twice during each microsecond, once during  $\phi 0$  and once during  $\phi 1$ , and VID7M remains low, enabling the clock continuously at 14M (Figure 11-23).

### ***Low-Resolution Display***

In the graphics modes, VA and VB are not used by the character generator, so the IOU uses lines SEGA and SEGB to transmit H0 and HIRES\*, as shown in Table 11-16.

**Table 11-16.** *Character-Generator Control Signals*

Display Mode	SEGA	SEGB	SEGC
Text	VA	VB	VC
Graphics	H0	HIRES*	VC

The low-resolution graphics display uses VC to divide the eight display lines corresponding to a row of characters into two groups of four lines each. Each row of data bytes is addressed eight times, the same as in text mode, but each byte is interpreted as two nibbles. Each nibble selects one of sixteen colors. During the upper four of the eight display lines, VC is low and the low-order nibble determines the color. During the lower four display lines, VC is high and the high-order nibble determines the color.

The bit patterns that produce the low-resolution colors are read from the character-generator ROM in the same way the bit patterns for characters are produced in text mode. The 74166 parallel-to-serial shift register converts the bit patterns to a serial bit stream for the video circuits (Figure 11-21).

The video signal generated by the Apple IIc includes a short burst of 3.58 MHz signal that is used by an NTSC color monitor or color TV set to generate a reference 3.58 MHz color signal. The Apple IIc's video signal produces color by interacting with this 3.58 MHz signal inside the monitor or TV set. Different bit patterns produce different colors by changing the duty cycles and delays of the bit stream relative to the 3.58 MHz color signal. To produce the small delays required for so many different colors, the shift register runs at 14 MHz and shifts out 14 bits during each cycle of the 1 MHz data clock. To generate a stream of fourteen bits from each eight-bit pattern read from the ROM, the output of the shift register is connected back to the register's serial input to repeat the same eight bits; the last two bits are ignored the second time around.

Each bit pattern is output for the same amount of time as a character: 1.02 microseconds. Because that is exactly enough time for three and a half cycles of the 3.58 MHz color signal, the phase relationship between the bit patterns and the signal

changes by a half cycle for each successive pattern. To compensate for this, the character generator puts out one of two different bit patterns for each nibble, depending on the state of H0, the low-order bit of the horizontal counter.

---

### ***High-Resolution Display***

The high-resolution graphics pages begin at memory locations \$2000 and \$4000 (decimal 8192 and 16384). These page addresses are selected by address bits A13 and A14. In high-resolution mode, these address bits are controlled by PAGE2 and 80STORE, the signals controlled by the display-page (PAGE2) and 80-column-video (80COL) soft switches. As in text mode, 80STORE inhibits addressing of the second page because there is only one page of 80-column text available for mixed mode.

In high-resolution graphics mode, the display data are still stored in blocks like the one shown in Figure 11-20, but there are eight of these blocks. As Tables 11-14 and 11-15 show, vertical counts VA, VB, and VC are used for address bits A10, A11, and A12, which address eight blocks of 1024 bytes each. Remember that in the display VA, VB, and VC count adjacent horizontal lines in groups of eight. This addressing scheme maps each of those lines into a different 1024-byte block.

It might help to think of this scheme as a kind of eight-way multiplexer: it's as if eight text displays were combined to produce a single high-resolution display, with each text display providing one line of dots in turn, instead of a row of characters.

The high-resolution bit patterns are produced by the character-generator ROM. In this mode, the bit patterns simply reproduce the seven bits of display data. The low-order six bits of data reach the ROM via the video data bus VID0-VID5. The IOU sends the other two data bits to the ROM via RA9 and RA10.

The high-resolution colors described in Chapter 2 are produced by the interaction between the video signal the bit patterns generate and the 3.58 MHz color signal generated inside the monitor or TV set. The high-resolution bit patterns are always shifted out at 7 MHz, so each dot corresponds to a half-cycle of the 3.58 MHz color signal. Any part of the video signal that produces a single white dot between two black dots, or vice

versa, is effectively a short burst of 3.58 MHz and is therefore displayed as color. In other words, a bit pattern consisting of alternating ones and zeros gets displayed as a line of color. The high-resolution graphics subroutines produce the appropriate bit patterns by masking the data bits with alternating ones and zeros.

To produce different colors, the bit patterns must have different phase relationships to the 3.58 MHz color signal. If alternating ones and zeros produce a certain color, say green, then reversing the pattern to zeros and ones will produce the complementary color, purple. As in the low-resolution mode, each bit pattern corresponds to three and a half cycles of the color signal, so the phase relationship between the data bits and the color signal changes by a half cycle for each successive byte of data. Here, however, the bit patterns produced by the hardware are the same for adjacent bytes; the color compensation is performed by the high-resolution software, which uses different color masks for data being displayed in even and odd columns.

To produce other colors, bit patterns must have other timing relationships to the 3.58 MHz color signal. In high-resolution mode, the Apple IIc produces two more colors by delaying the output of the shift register by half a dot (70 ns), depending on the high-order bit of the data byte being displayed. (The high-order bit doesn't actually get displayed as a dot, because at 7 MHz there is only time to shift out seven of the eight bits.)

As each byte of data is sent from the character generator to the shift register, high-order data bit D7 is also sent to the TMG. If D7 is off, the TMG transmits shift-register timing signals LDPS\* and VID7M normally. If D7 is on, the TMG delays LDPS\* and VID7M by 70 nanoseconds, the time corresponding to half a dot. The bit pattern that formerly produced green now produces orange; the pattern for purple now produces blue.

**A Note About Timing:** For 80-column text, the shift register is clocked at twice normal speed. When 80-column text is used with graphics in mixed mode, the TMG controls shift-register timing signals LDPS\* and VID7M so that the graphics portion of the display works correctly even when the text window is in 80-column mode.



---

### ***Double-High-Resolution Display***

Double-high-resolution graphics mode displays two bytes in the time normally required for one, but it uses High-resolution Graphics Pages 1 and 1X instead of Text and Low-resolution Pages 1 and 1X.

**Note:** There is a second pair, HRP2 and HRP2X, which can be used to display a second double-high-resolution page.

Double-high-resolution graphics mode displays each pair of data bytes as 14 adjacent dots, seven from each byte. The high-order bit (color-select bit) of each byte is ignored. The auxiliary-memory byte is displayed first, so data from auxiliary memory appears in columns 0-6, 14-20, and so on, up to columns 547-552. Data from main memory appears in columns 7-13, 21-27, and so on, up to 553-559.

As in 80-column text, there are twice as many dots across the display screen, so the dots are only half as wide. On a TV set or low-bandwidth (less than 14 MHz) monitor, single dots will be dimmer than normal.

**RGB** stands for red, green and blue and identifies a type of color monitor that uses independent inputs for the three primary colors.

**Note:** Except for some expensive RGB-type color monitors, any video monitor with a bandwidth as high as 14 MHz will be a monochrome monitor. Monochrome means one color: a monochrome video monitor can have a screen color of white, green, orange, or any other single color.

The main memory and auxiliary memory are connected to the address bus in parallel, so both are activated during the display cycle. The rising edge of  $\phi 0$  clocks a byte of main memory data into the video latch, and a byte of auxiliary memory data into the 80 latch (Figure 11-21).

Phi 1 enables output from the (auxiliary) 80 latch, and  $\phi 0$  enables output from the (main) video latch. Output from both latches goes to CHARGEN, where GR and SEGB\* select high-resolution graphics. LDPS operates at 2 MHz in this mode, alternately gating the auxiliary byte and main byte into the parallel-to-serial shift register. VID7M is active (kept true) for double-high-resolution display mode, so when it is ANDed with 14M, the result is still 14M. The 14M serial clock signal gates shift register output to VID, the video display hybrid circuit, for output to the display device.

For further information about double-high-resolution graphics display, refer to the Bibliography.

**Figure 11-22.** 7 MHz Video Timing Signals (40-Column, Low-Resolution and High-Resolution Display)

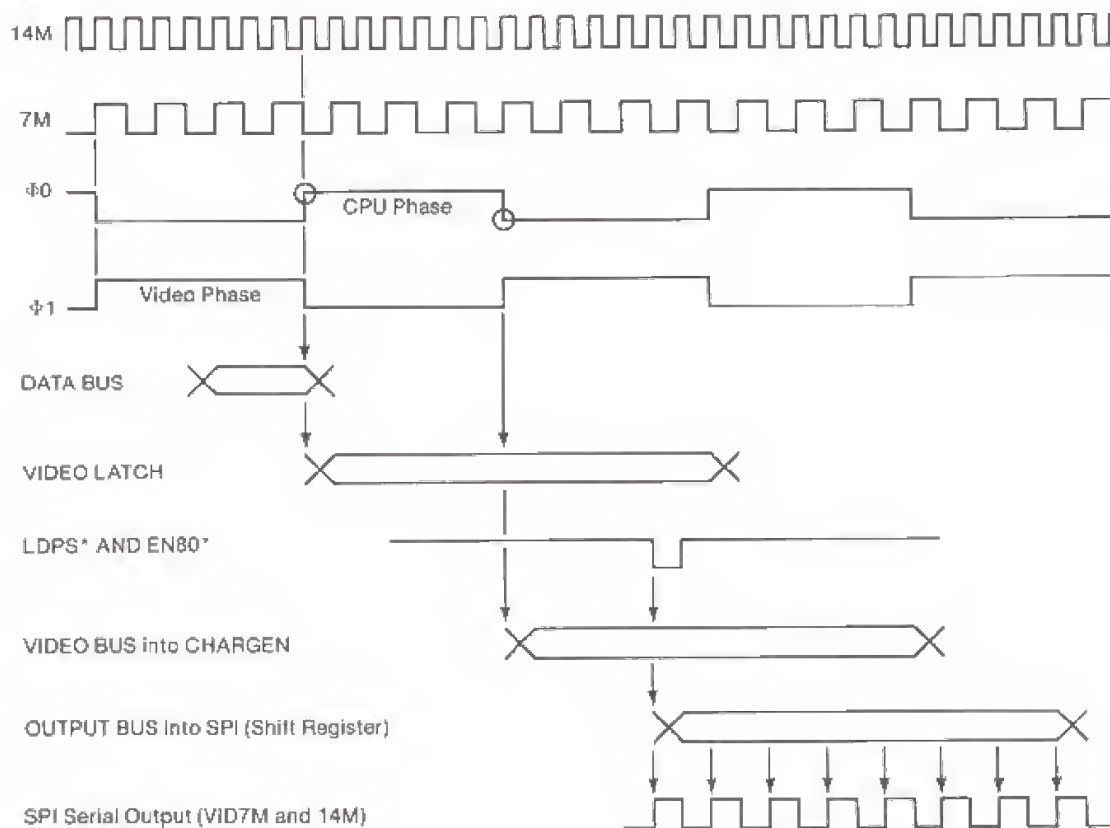
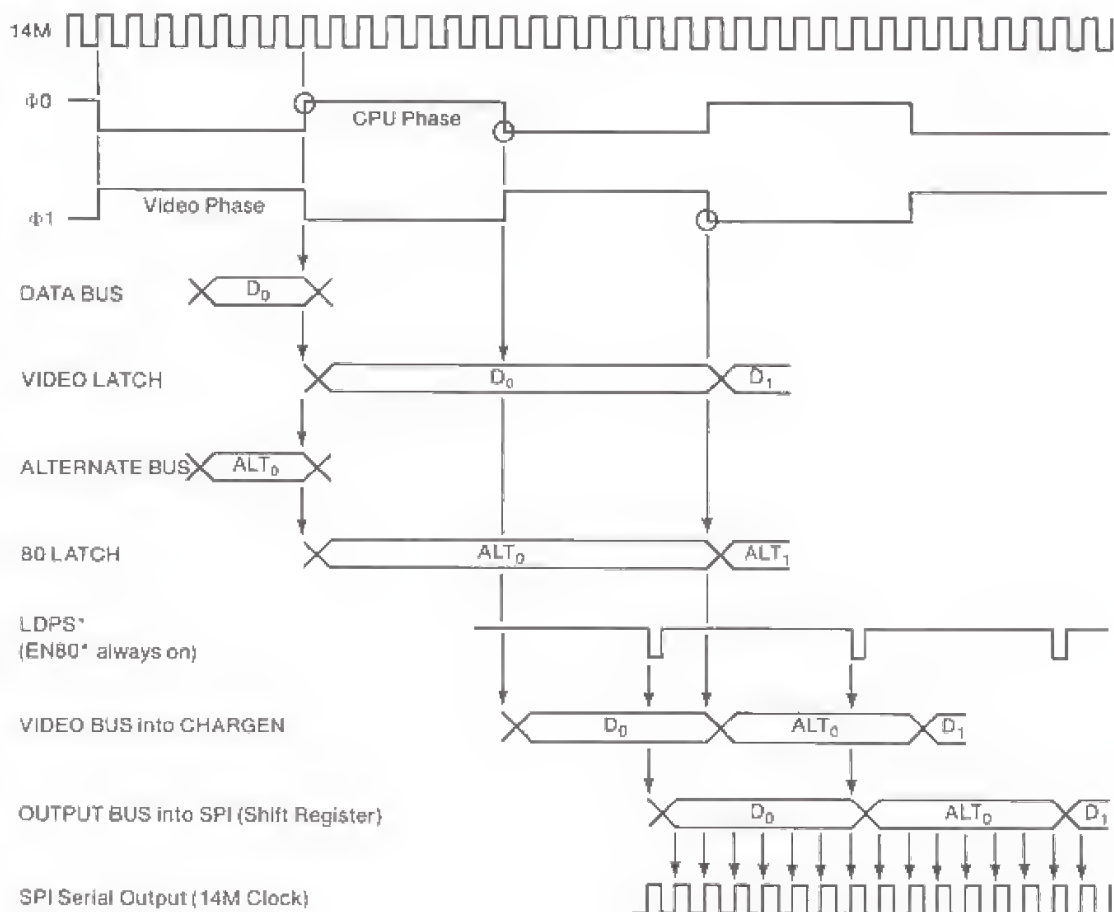




Figure 11-23. 14 MHz Video Timing Signals (80-Column and Double-High-Resolution Display)



VID is a video-amplifier hybrid circuit.

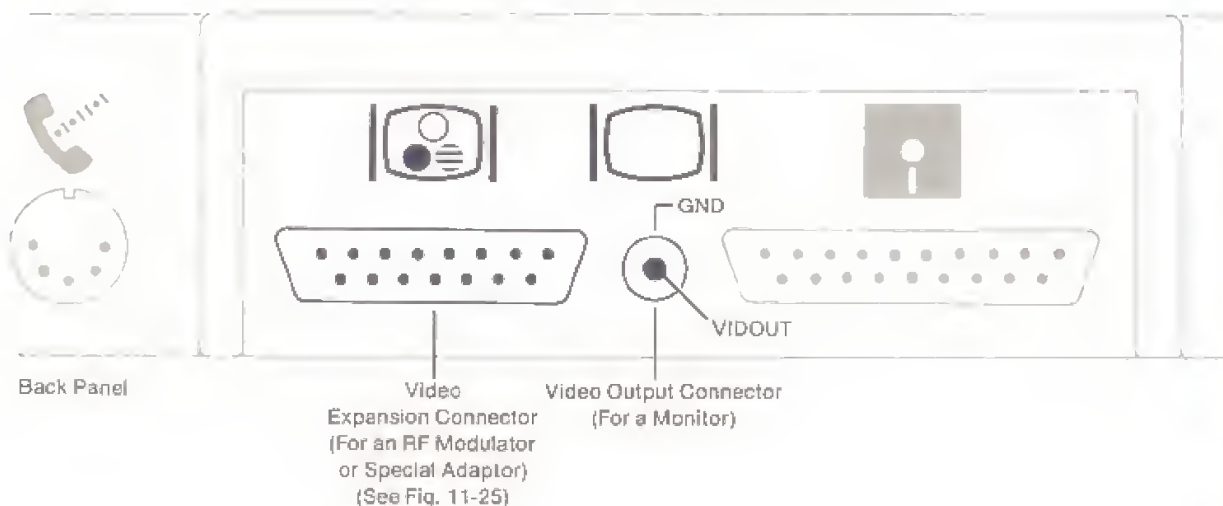
### 11.9.5 Video Output Signals

The stream of video data generated by the display circuits described above goes to a hybrid circuit (VID) that adjusts the signals to the proper amplitudes and conditions the color burst.

The resulting video signal is an NTSC-compatible composite-video signal that can be displayed on a standard video monitor. The signal is similar to the EIA (Electronic Industries Association) standard positive composite video. This signal is available in two places in the Apple IIc (Figure 11-24):

- at the phono jack on the back of the Apple IIc
- at the video expansion connector (pin 12) on the back panel. (Table 11-17).

*Figure 11-24. Video Output Back Panel Connectors*



### Monitor Output

The sleeve of the phono jack at the center of the Apple IIc back panel is connected to ground and the tip is connected to the video output through a resistor network that attenuates it to about 1 volt and matches its impedance to 75 ohms. This arrangement is suitable for most video monitors.

---

### Video Expansion Output

The back panel of the Apple IIc has a DB-15 connector for sophisticated video interfaces external to the computer. Figure 11-25 shows the pin assignments for this connector; Table 11-17 describes the signals.

In Table 11-17, the column labeled *Deriv* indicates what clock signals the video signals are derived from. LDPS, CREF and PRAS have a maximum delay of 30 ns from the appropriate 14 MHz rising edge. SEROUT is clocked out of a 74LS166 by the rising edge of 14M and has a maximum delay of 35 ns. VIDD7 is driven from a 74LS374 and has a maximum delay of 28 ns from the rising and (if 80-column) falling edges of  $\phi 1$ .

To align CREF so it is in the same phase at the beginning of every line, certain clock signals must be *stretched*. This stretch is for one 7M cycle (140 ns), and occurs at the end of each video line. All timing signals except 14M, 7M and CREF are stretched.

---

#### Warning

*The signals at the DB-15 on the Apple IIc are not the same as those at the DB-15 on the Apple III. Do not attempt to plug a cable intended for one into the other.*

---

---

#### Warning

*Several of these signals, such as 14 MHz, must be buffered within about four inches (10 cm) of the back panel connector—preferably inside a container directly connected to the back panel. For technical information, contact Apple Technical Support.*

---

*Figure 11-25. The Video Expansion Connector Pinouts*



Pin	Signal	Pin	Signal
1	TEXT	9	PRAS*
2	14MHz	10	GR
3	SYNC*	11	SEROUT*
4	SEGB	12	NTSC
5	1VSOUND	13	GND
6	LDPS*	14	VIDD7
7	WNDW*	15	CREF
8	+12V		

**Table 11-17. The Video Expansion Connector Signals**

Pin	Deriv	Name	Description
1	$\phi 0$	TEXT	Video text signal from TMG; set to inverse of GR, except in double-high-resolution mode
2		14MHz	14 MHz master timing signal from the system oscillator
3	Q3	SYNC*	Display horizontal and vertical synchronization signal from IOU pin 39
4	PRAS	SEGB	Display vertical counter bit from IOU pin 4; in text mode indicates second low-order vertical counter; in graphics mode indicates low-resolution
5		1VSOUND	One-volt sound signal from pin 5 of the audio hybrid circuit (AUD)
6	14MHz	LDPS*	Video shift-register load enable from pin 12 of TMG
7	PRAS	WNDW*	Active area display blanking; includes both horizontal and vertical blanking
8		+12 V	Regulated +12 volts DC.; can drive 350 mA
9	14MHz	PRAS*	RAM row-address strobe from TMG pin 19
10	PRAS	GR	Graphics mode enable from IOU pin 2
11	14MHz	SEROUT*	Serialized character-generator output from pin 1 of the 74LS166 shift register
12		NTSC	Composite NTSC video signal from VID hybrid chip
13		GND	Ground reference and supply
14	$\phi 0$	VIDD7	From 74LS374 video latch; causes half-dot shift if high
15	14MHz	CREF	Color reference signal from TMG pin 3; 3.58 MHz

---

**Warning**

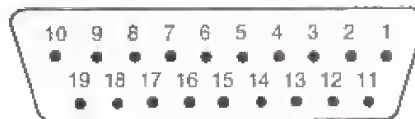
*Use caution—The maximum allowable current drain of +12 V regulated power at the video expansion connector is 300 milliamps. If the external device draws more than this, it can damage the computer or cause the power supply to shut down.*

---

## 11.10 Disk I/O

Disk I/O—for both the built-in and the external drive—is supported by the IWM disk controller unit. The external drive is attached via a DB-19 connector. Figure 11-26 shows this connector. Table 11-18 describes the pin assignments. Supply voltages come from the power supply; all other signals come from the IWM, described in section 11.5.5.

*Figure 11-26. Disk Drive Connector*



Pin	Signal	Pin	Signal
1,2,3,4	GND	13	SEEKPH2
5	-12V	14	SEEKPH3
6	+5V	15	WRREQ*
7,8	+12V	16	N.C.
9	EXTINT*	17	DR2*
10	WRPROT	18	RDDATA
11	SEEKPH0	19	WRDATA
12	SEEKPH1		

---

**Warning**

*The power available at this connector is for a Disk II or similar drive only. Do not use power from the external disk connector for any other purpose—you may damage the internal voltage converter. To derive external power for an attached device, use one of the other connectors and observe the current limits given in this manual.*

---



**Table 11-18. Disk Drive Connector Signals.** \*\*Refer to the warning preceding this table.

Connector Pin Number	Name	Description
1,2,3,4	GND	Ground reference and supply
6	+5 V	+5 volt supply**
7,8	+12	+12 volt supply**
9	EXTINT*	External interrupt
10	WRPROT	Write-protect input
11-14	PH0-4	Motor phase 0-4 output
15	WRREQ*	Write Request
17	DR1*	Drive 1 select
18	RDDATA	Read data input
19	WRDATA	Write data output

## 11.11 Serial I/O

Apple IIc has built into it two 6551 Asynchronous Communication Interface Adapters (ACIA) and supporting input and output buffers for full-duplex serial communication. Figure 11-27 is a block diagram of the Apple IIc serial ports. ACIA outputs are buffered by a 1448 quad line driver. Similarly, ACIA inputs are buffered by a 1489 quad line receiver.

The RS-232 signals are defined in the Glossary.

Figure 11-28 is a detail block diagram of the 6551 ACIA. The registers are described in sections 11.11.1 through 11.11.4.

Figure 11-27. Serial Port Circuits

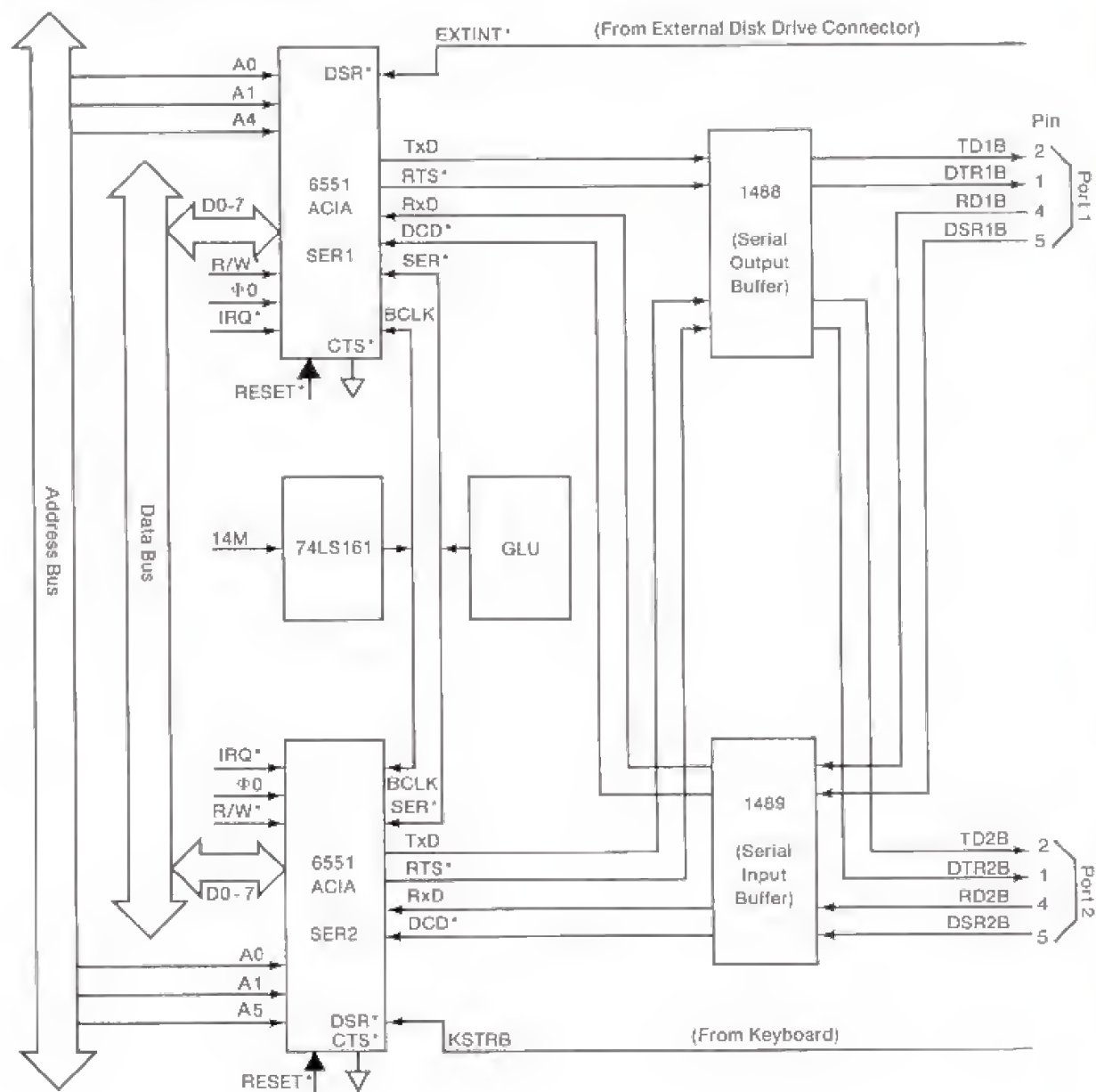
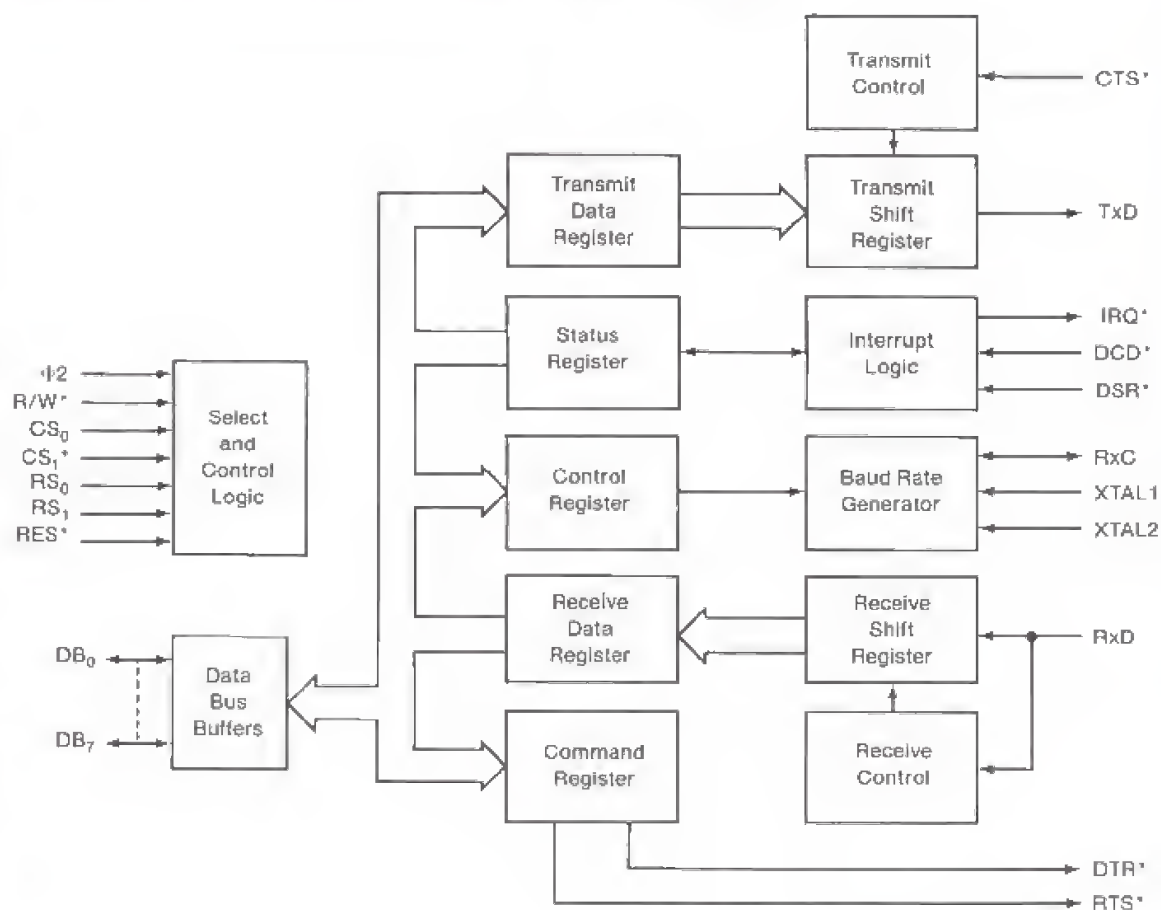


Figure 11-28. 6551 ACIA Block Diagram. Copyright 1978, Synertek Inc.  
Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.



The 6551 pin assignments are shown in Figure 11-29 and described in Table 11-19. Note that the two 6551s are not used in exactly the same way—each one supports a different set of interrupts.

Port 1 reads external interrupts (EXTINT\*) on its Data Set Ready (DSR) pin. This input is tied to +5 V through a 3.3 Kohm pullup resistor.

Figure 11-29. The 6551 Pinouts

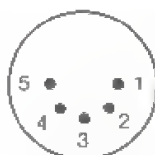
GND	1	28	R/W*
A5	2	27	Φ0
SER*	3	26	IRQ*
RESET*	4	25	D7
N.C.	5	24	D6
BCLK	6	23	D5
N.C.	7	22	D4
RTS*	8	21	D3
GND	9	20	D2
TxD	10	19	D1
N.C.	11	18	D0
RxD	12	17	OSR*
A0	13	16	DCD*
A1	14	15	+5V

Table 11-19. The 6551 Signal Descriptions

Pin Number	Name	Description
1	GND	Power and signal common ground
2	A4	Address line 4 to select serial port 1
	A5	Address line 5 to select serial port 2
3	SER*	Serial device select from GLU
4	RESET*	Resets both serial ports
5	-	No connection
6	BCLK	Baud rate clock from GLU
7	-	No connection
8	RTS*	Request to Send output
9	CTS*	Clear to Send input
10	TXD	Transmit Data output
11	-	No connection
12	RXD	Receive Data Input
13,14	A0,A1	Address lines 0 and 1
15	+5 V	+5 volt supply
16	DSR	Data Set Ready input
17	EXTINT*	External interrupt (port 1 ACIA)
	KSTRB	Keyboard strobe input (port 2 ACIA; Appendix E)
18-25	D0-D7	Eight-bit data bus
26	IRQ*	Interrupt Request Input
27	Φ0	Phase 0 clock pulse
28	R/W*	Read/write select input

The back panel connectors for both serial ports are 5-pin DIN jacks. The pin assignments are shown in Figure 11-30 and described in Table 11-20.

*Figure 11-30. Serial Port Connectors*



Pin	Port 1	Port 2
1	DTR1B	DTR2B
2	TD1B	TD2B
3	GND	GND
4	RD1B	RD2B
5	DSR1B	DSR2B

*Table 11-20. Serial Port Connector Signals*

Pin Number	Name	Description
1	DTR1B	Data Terminal Ready output
	DTR2B	
2	TD1B	Transmit Data output
	TD2B	
3	GND	Power and signal common
4	RD1B	Read Data input
	RD2B	
5	DSR1B	Data Set Ready input
	DSR2B	

---

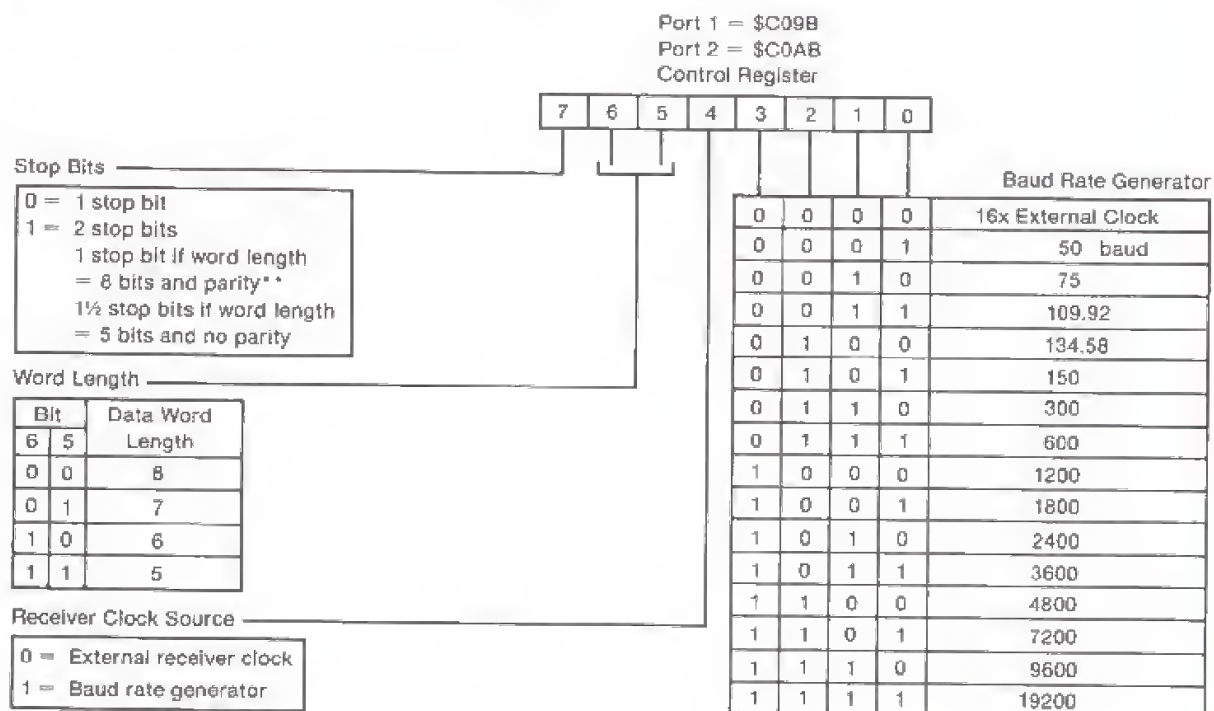
### 11.11.1 ACIA Control Register

Figure 11-31 shows the bit assignments for the ACIA Control Register, which the hardware locates at address \$C09B for serial port 1, and \$C0AB for serial port 2. This register determines the number of data and stop bits the ACIA will transmit and receive, and the clock source and baud rate to use for data transfer.

The receiver clock source is derived from the Apple IIc's TMG chip; the resulting baud rates are equal to or up to 2% lower than the nominal rate. (The EIA standard allows plus or minus 2% variation.) If an Apple IIc serial port is used with a modem that is 2% above the nominal rate, framing errors can occur, especially at 1200 baud and above, when using eight data bits. It may be necessary to select a lower baud rate for 8-bit binary data transfers.



Figure 11-31. ACIA Control Register. Copyright 1978, Synertek Inc.  
Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.



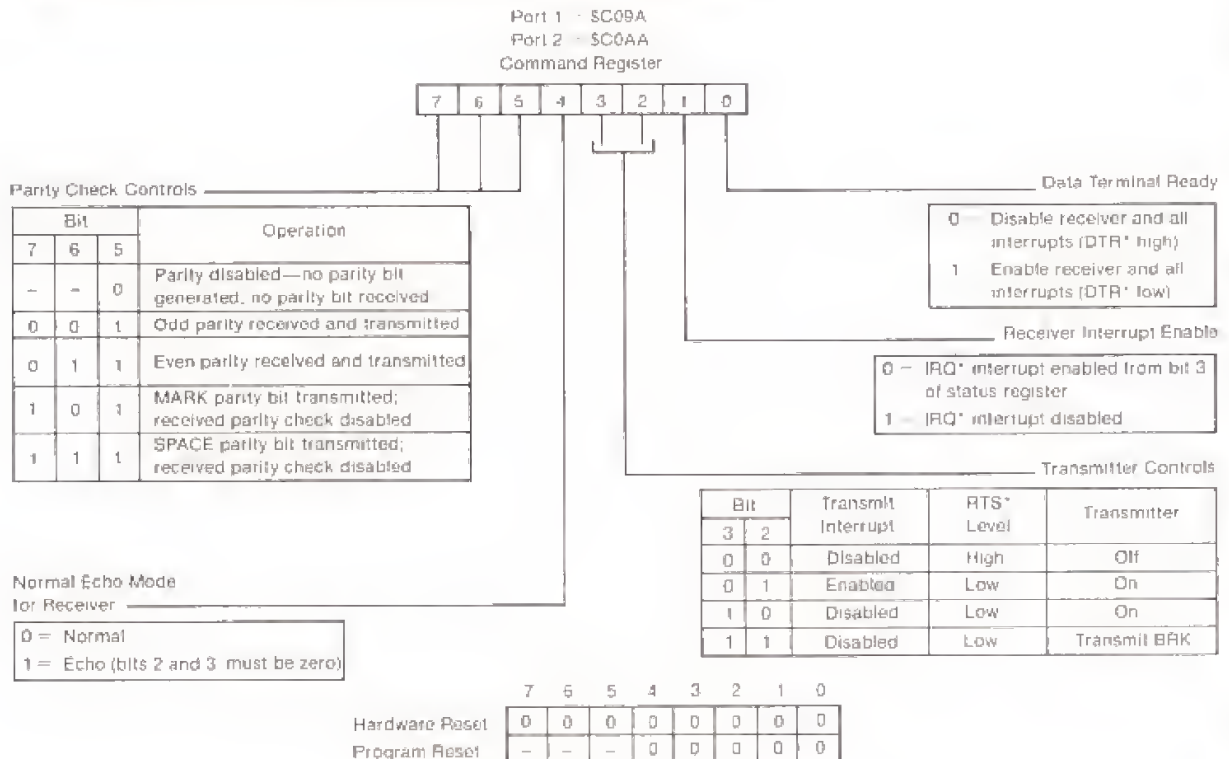
\*\*This allows for 9-bit transmission (8 data plus parity).

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	-	-	-	-	-	-	-	-

### 11.11.2 ACIA Command Register

Figure 11-32 shows the bit assignments for the ACIA Command register, which the hardware locates at address \$C09A for serial port 1, and at \$C0AA for serial port 2. This register controls specific transmit and receive functions: parity checking, echoing input to output, allowing transmit and receive interrupts, and setting levels for Data Terminal Ready and Request to Send.

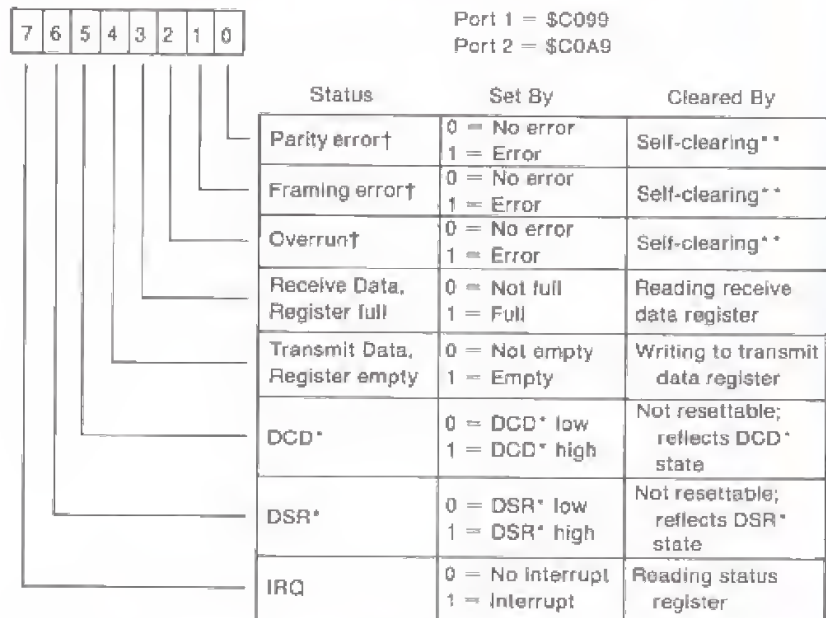
**Figure 11-32.** ACIA Command Register. Copyright 1978, Synertek Inc.  
Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.



### 11.11.3 ACIA Status Register

Figure 11-33 shows the bit assignments for the ACIA Status Register, which is hard-wired to address \$C099 for serial port 1, and \$C0A9 for serial port 2. This register reports the condition of the transmit/receive register, errors detected during data transfer, and the level of the Data Carrier Detect, Data Set Ready and interrupt request lines.

**Figure 11-33. ACIA Status Register.** Copyright 1978, Synertek Inc.  
Used by permission of Synertek Inc., 3001 Stender, Santa Clara, CA 95052.



† No interrupt generated for these conditions.

\*\* Cleared automatically after a read of RDR and the next error-free receipt of data.

	7	6	5	4	3	2	1	0
Hardware Reset	0	0	0	0	0	0	0	0
Program Reset	-	-	-	-	-	-	-	-

---

#### **11.11.4 ACIA Transmit/Receive Register**

Each ACIA uses the same address—\$C098 for serial port 1, \$C0A8 for serial port 2—as temporary data storage for both transmission and reception of data.

When the register is used for transmitting data, bit 0 is the leading bit to be transmitted; unused data bits are the high-order bits, which are ignored.

When the register is used for receiving data, bit 0 is the first bit received; unused data bits are the high-order bits, which are set to 0. Parity bits never appear in the receive data register; they are stripped off after being used for external parity checking.

---

### **11.12 Mouse Input**

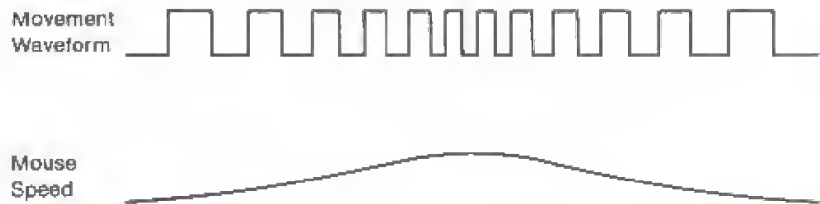
The mouse is a hand-held X-Y pointing device that can be rolled along a flat surface. It has an attached pushbutton. This section describes how mouse movement and direction can be detected and interpreted.

A mouse has a ball inside its housing that protrudes a small distance so that its turning corresponds to mouse movements across a tabletop. Two wheels inside the housing, set at 90-degree angles to each other, follow movements of the ball; this causes two disks to rotate. The disks have rectangular holes arranged near their edges, making them resemble circular slide mounts used with stereoscopic slide viewers.

The light from a tiny infrared emitter reaches a photoreceptor whenever one of the holes on the disk lies between them. An internal circuit in the mouse causes the resulting voltage to swing quickly to a 1 or a 0 value as soon as a certain threshold is crossed. The result is something approximating a square wave (Figure 11-34) that varies directly with the speed of mouse movement. One of these indicates the X component (X0) of mouse movement; the other, the Y component (Y0).

Under program control, either the rising edge or the falling edge of each square wave can cause an interrupt, which the firmware handles by updating a counter. However, the program needs to know whether to add or to subtract 1 from a counter; that is, it needs to know the direction of X or Y movement.

Figure 11-34. Sample Mouse Waveform



There is a second infrared emitter/photoreceptor pair almost 180 degrees opposite the first pair for each disk. These pairs are positioned in such a way that the square waves they generate are approximately a quarter-wave offset from their respective movement waves (Figure 11-35). These waveforms are called **X1** (X direction) and **Y1** (Y direction).

When a rising edge of **X0** causes an interrupt, a mouse-driver program can immediately check whether **X1** is 0 (indicating a movement to the right) or 1 (indicating a movement to the left). Similarly, the mouse driver can read **Y1** immediately after a **Y0** interrupt to determine whether the mouse moved up or down one count along the Y axis.

Figure 11-35. Mouse Movement and Direction Waveforms

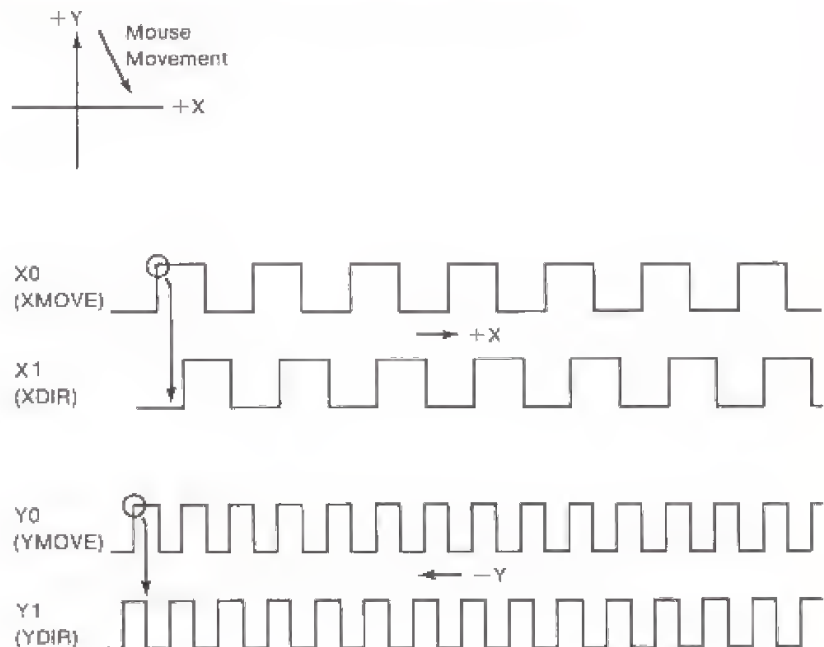


Figure 11-36 shows the pin assignments for the mouse DB-9 connector on the back panel. Table 11-21 gives the signal names and descriptions.

*Figure 11-36. Mouse Connector*



Pin	Signal
1	MOUSEID*
2	+5V
3	GND
4	XDIR
5	XMOVE
6	(N.C.)
7	MSW*
8	YDIR
9	YMOVE

*Table 11-21. Mouse Connector*

DB-9 Pin Number	Signal Name	Description
1	MOUSEID*	Mouse identifier; when active, disables NE556 hand controller timer.
2	+5V	Total current drain from this pin must not exceed 100 mA.
3	GND	System ground
4	X1	Mouse X-direction indicator
5	X0	Mouse X-movement interrupt
6		Mouse button
7	MSW*	Mouse button
8	Y1	Mouse Y-direction indicator
9	Y0	Mouse Y movement interrupt



Figure 11-37 shows the mouse and hand control circuitry with the mouse circuits highlighted. Figure 11-38 illustrates the values of the mouse-button circuit when the button is pressed or not pressed. Pressing the button disables the NE556 by pulling the reset comparator threshold value up so that it cannot reset the flip flop. As a result the mouse-button input value remains at a TTL level.

Figure 11-37. Mouse Circuits

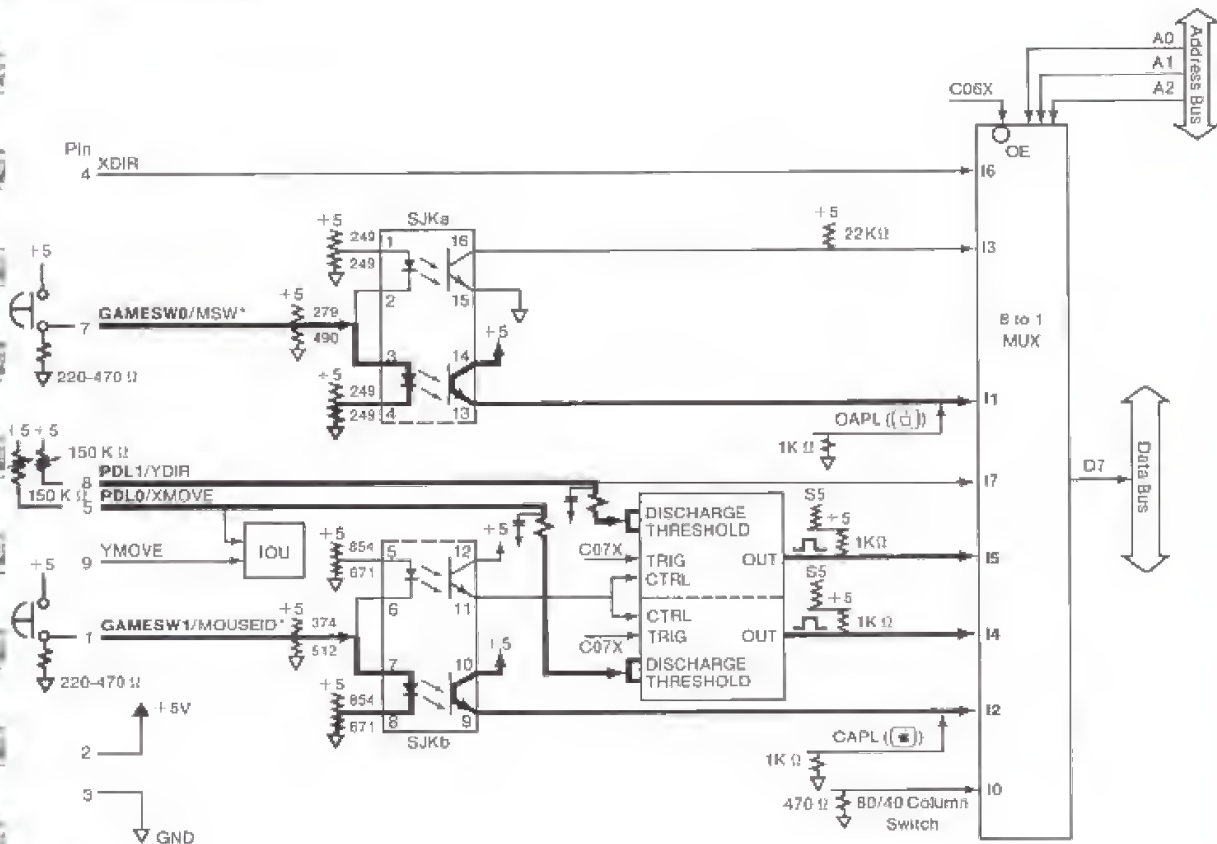
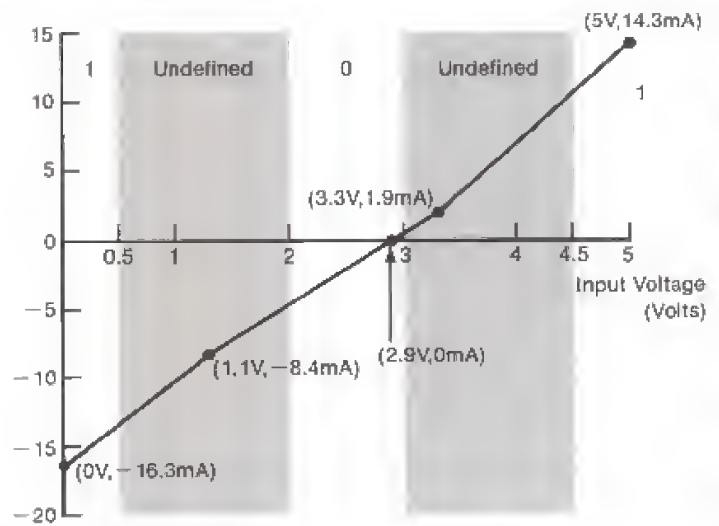
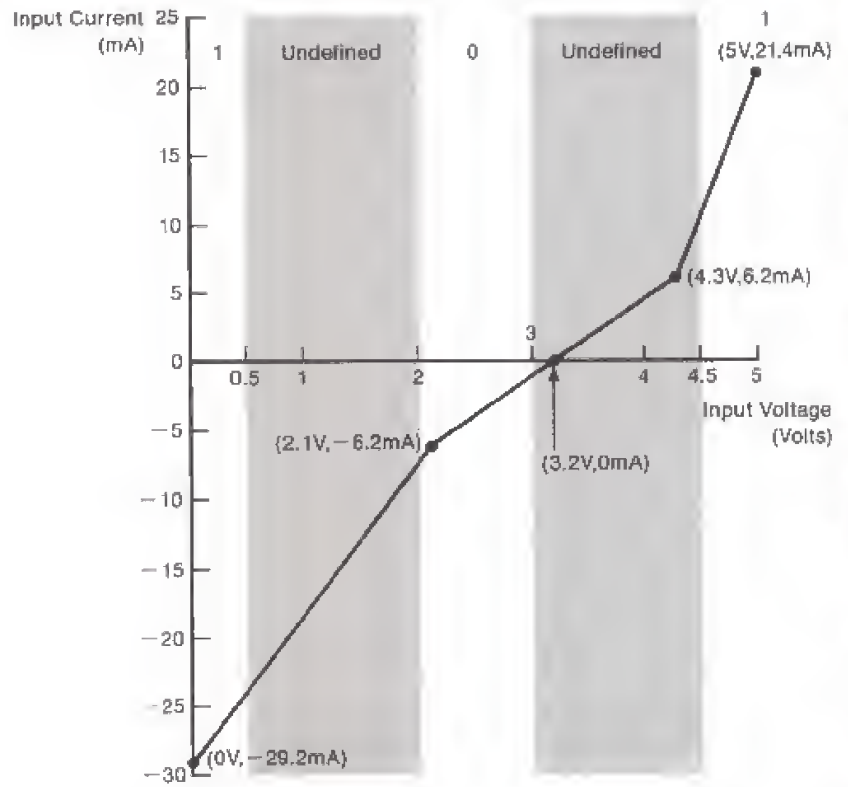


Figure 11-38. Mouse Button Signals



## 11.13 Hand Controller Input

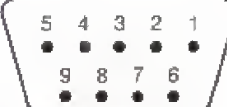
Several input signals that are individually controlled via soft switches are collectively referred to as the **hand controller (game) signals**. These signals arrive in the Apple IIc via the same DB-9 connector as the one used for the mouse (section 11.12), but the Apple IIc interprets these signals differently.

The DB-9 connector pin assignments and signal descriptions, as used for hand control input, appear in Figure 11-39 and Table 11-22.

Even though they are normally used for hand controls, these signals can be used for other simple I/O applications. There are two one-bit switch inputs, labeled SW0 and SW1, and two analog inputs, called **paddles** and labeled PDL0 and PDL1. Figure 11-40 shows how to connect the one-bit switch inputs for compatibility with all other Apple II series computers.

The switch inputs are multiplexed by a 74LS251 8-to-1 multiplexer enabled by the C06X\* signal from the MMU. Depending on the low-order address, the appropriate game input is connected to bit 7 of the data bus. Figure 11-41 shows the mouse and hand control circuitry with the hand control circuits highlighted. Figure 11-42 illustrates the values of the hand-control switch inputs when the switch is open or closed.

Figure 11-39. Hand Controller Connector



Pin	Signal
1	GAMESW1
2	+5V
3	GND
4	Not used for hand controls
5	PDL0
6	(N.C.)
7	GAMESW0
8	PDL1
9	Not used for hand controls

**Table 11-22. Hand Control Connector Signals**

Connector Pin Number	Signal Name	Description
1	GAMESW1	Switch input 1 (sometimes called <b>paddle button 1</b> )
2	+5V	+5 V power supply. Total current drain from this pin must not exceed 100 mA.
3	GND	System ground
4,9	-	Not used for hand controls
5,8	PDL0 and PDL1	Hand control inputs. Each of these must be connected to a 150 Kohm variable resistor connected to +5V.
6	N.C.	Not connected
7	GAMESW0	Switch Input 0 (sometimes called <b>paddle button 0</b> )

**Figure 11-40. How to Connect Switch Inputs**

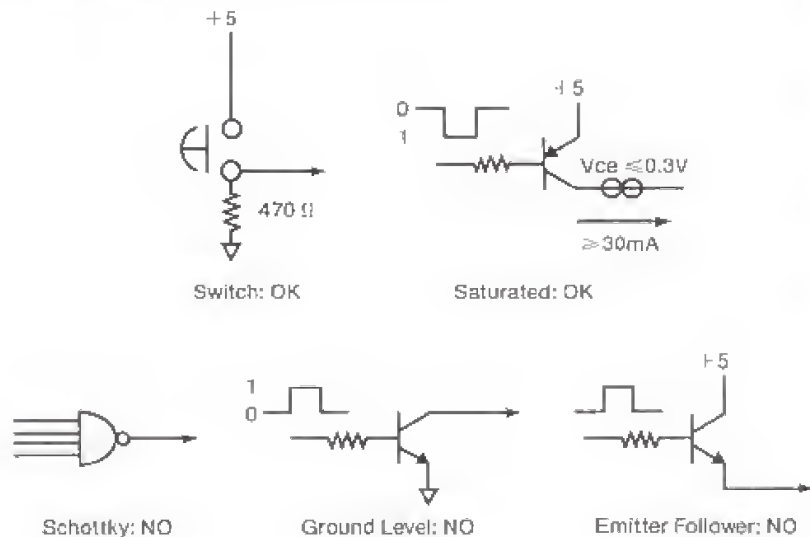


Figure 11-41. Hand Control Circuits

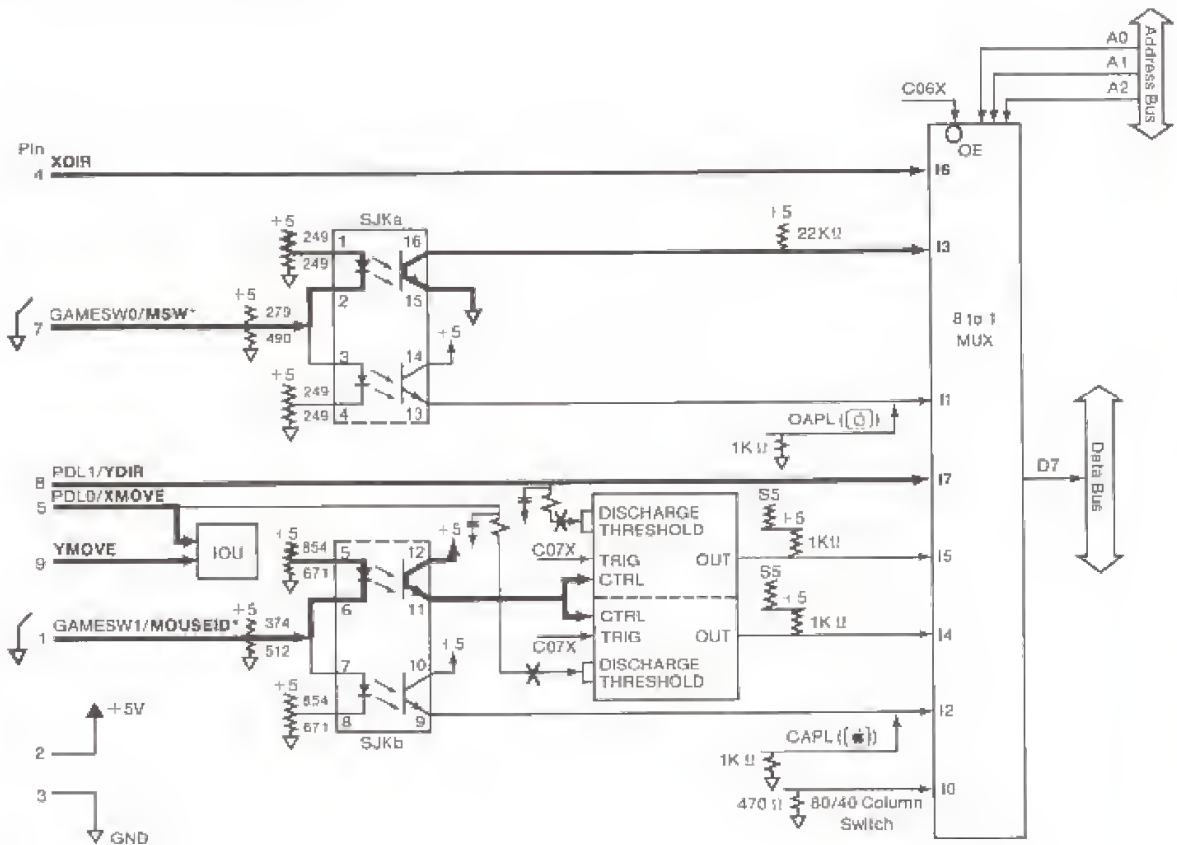
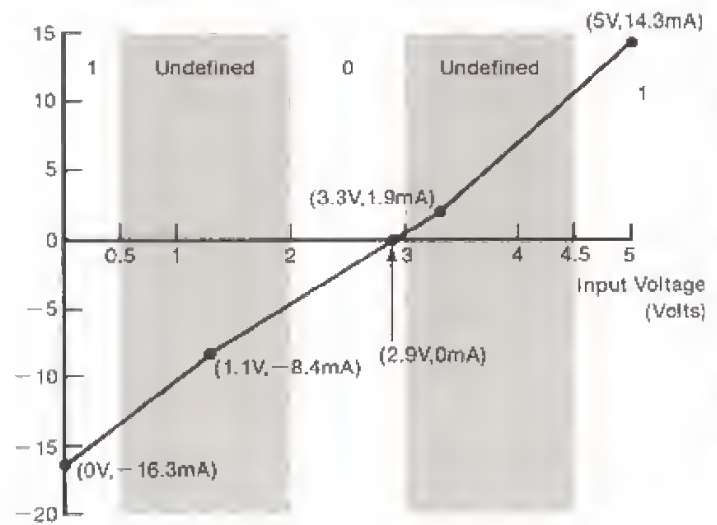
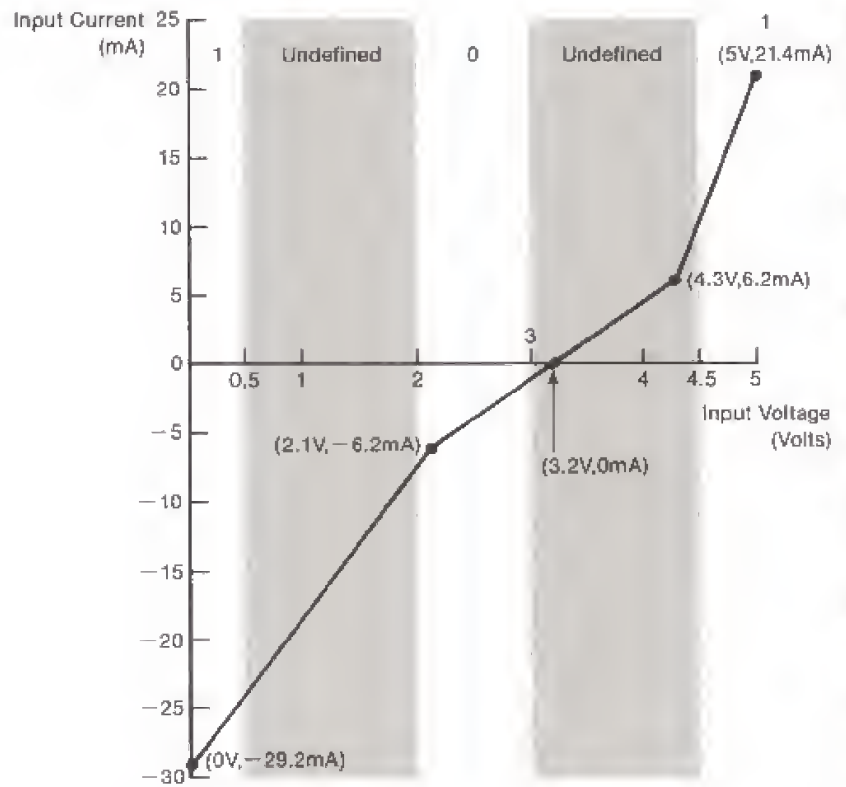


Figure 11-42. Hand Control Signals





The hand-control inputs are connected to the timing inputs of an NE556 dual analog timer. Addressing \$C07X sends a signal from MMU pin 22 that resets both timers and causes their outputs to go to 1 (high). A variable resistance of up to 150 Kohms connected between one of these inputs and the +5 V supply controls the charging time of one of the two 0.022 microfarad capacitors.

When the voltage on the capacitor passes a certain threshold, the output of the NE556 changes back to 0 (low). Programs can determine the setting of a variable resistor by resetting the timers and then counting time until the selected timer input changes from high to low. The resulting count is proportional to the resistance.



---

**Warning**

*The only way to ensure correct paddle values is to make sure the output of the paddle you intend to read is low before you trigger the timer. Triggering the timer starts the charging cycle for the capacitor in each paddle circuit; the cycle for one may not be completed by the time you have read the other. If you retrigger or read the other paddle too soon (that is, in less than 3 ms), you will get a false value for it.*

---

## **11.14 Schematic Diagrams**

The following pages contain schematic diagrams for the Apple IIc.

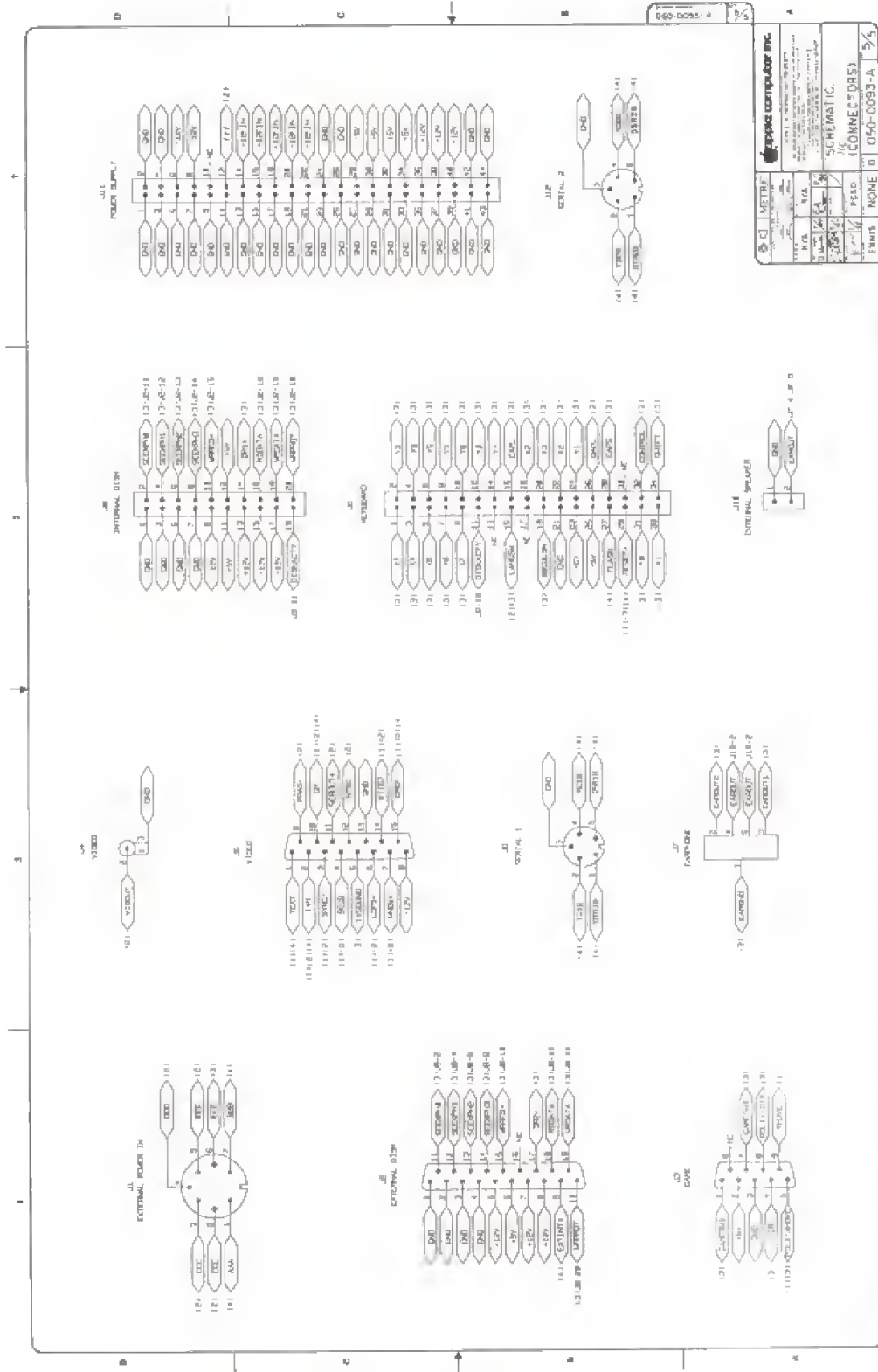
















# *Index*

References to entries in Volume 2 are in square brackets [ ].

### ***Cast of Characters***

\* (asterisk) 179  
 \ (backslash) 59  
 \_ (blinking underscore cursor) 154  
 > (greater than sign) 59  
 ? (question mark) 58, 59  
 ] (right bracket) 59

### **A**

A register 17  
 accumulator 17  
 ACIA 134, 148, 253-262, [63]  
     block diagram 255  
     interrupts [60]  
 address bus 12, 213  
 AKD 218-219  
 ALTCHAR 104-105, 218, [73]  
 alternate character set 68, [73]  
 ALTZP 25, 26, 216, [46]  
 analog inputs 176, [68]  
 annunciator outputs [76]  
 ANSI [84]  
 any-key-down 79, 229  
     flag [66]  
 Apple Extended 80-Column  
     Text Card [67, 74]  
 Apple Language Card [64]

Apple II series differences [60-78]  
 Apple IIc  
     block diagram 210  
     care of 205-206  
     differences from Apple IIe [61-78]  
     expansion 2  
 Apple IIe ROMs [72]  
 Applesoft & commands 52  
 Applesoft BASIC 59, [16-18, 40]  
     BASIC interpreter 24  
 Applesoft interpreter 21, 224-225  
 arithmetic, hexadecimal 193  
 ASCII [71, 83, 86-87]  
     character set 79, [97, 114-122]  
 assemblers 199  
 assembly language, and mouse 171  
 asterisk (\*) 179  
 automatic line feed 131, 145  
 automatic repeat 3  
 Autostart ROM [69]  
 auxiliary memory screen holes  
     135-136, 149-150  
     *See also* screen holes  
 auxiliary RAM 20  
 AUXMOVE *See* MOVEAUX  
 AY-3600-type keyboard decoder 229

## B

- B command 131, 144
- back panel 8, 9
- backslash (\) 59, 62
- backspace 62
- bank 25
- bank-switched memory 22, [64, 69]
- BANK2 216
- BASIC 130, 163, 175-177, 179, 180, 192, [114]
  - and assembly language support 171
  - and hand controls 173
  - and mouse 163, 172
- BASICS* disk [39, 69]
- baud rate 137, 258
- BCLK 256
- BELL 84
- BELL1 84
- BIT instruction [3]
- bits [103]
- blanking intervals 233
- blinking underscore cursor ( \_ ) 154
- block diagrams
  - ACIA 255
  - Apple IIc 210
- BREAK 132, 137, 145
- break instructions [48]
- BREAK signal [75]
- BRK 75, 189, [43]
- buffer 59
  - serial I/O [75]
- built-in diagnostics [62]
- built-in disk drive 8
- built-in self-tests [65]
- button interrupt mode 164, 167
- bypassing firmware [58-60]
- byte(s) [103, 104]
  - power-up 51

## C

- C06X 267
- C07X 217
- C3COUT1 55, 64

- C3KEYIN 55
- CALL statement 179
- Canadian keyboard [91]
- cancel line 62
- CAPS-LOCK** 4, 79, [84]
- card(s) [74, 75]
- care of computer 205-206
- carriage return 139, 152
- carrier 137
- CAS (column-address strobe) 228
- cassette input and output [67-68, 77]
- certifications [99]
- CH (cursor horizontal) 63
- changing memory contents 184
- changing registers 190
- character(s)
  - flashing 68
  - generator 241
  - inverse 68
  - normal 68
  - sets [71, 73]
- chips, custom [78]
- clamping boundaries 171
- CLAMP MOUSE 168
- CLEAR MOUSE 168
- CLEOLZ 116
- clock 211
  - master 213
  - system 213
- CLREOL 116
- CLREOP 116
- CLRSCR 117
- CLRTOP 117
- code conversions [114-122]
- cold-start procedure 49, 50
- colors
  - high-resolution 243
  - low-resolution 242, [63]
- command character 146, [75]
- command register 134, 148, 260
- Communication Card [74]
- communication port 141
- comparing data in memory 188-189

- connector(s)
    - back panel 8-9
    - game [76]
    - power 207
    - serial port 257
  - CONTINUE BASIC command 192
  - CONTROL** 4, 79, 229
    - transferring 42-43
  - control characters 64
  - control register 134, 148, 258-259
  - CONTROL-A, as command character 143
  - CONTROL-C** [53]
  - CONTROL-H** 62
  - CONTROL-I, as command character 130, 132
  - CONTROL-K, as command character 193
  - CONTROL-P** 56, 126, 142
    - as command character 193
  - CONTROL-R** 155
  - CONTROL-S** [53]
  - CONTROL-T** 156, 159
  - CONTROL-X** 62
  - CONTROL-Y** 197
    - commands 52
  - CONTROL RESET 50
    - conversion, number [106]
  - COUT 55, 117, 191
  - COUT1 55, 68, 117
  - CP/M [40]
  - CPU *See* 65C02
  - CR *See* carriage return
  - CREF 220, 221, 251
  - CROUT 117
  - CROUT1 117
  - CSW 56, 70, 104
  - cursor 58, 130, 143, 193
    - blinking underscore ( \_ ) 154
    - flashing checkerboard 55
    - flashing question mark 130, 143
    - inverse solid 55
  - custom chips [78]
  - custom integrated circuits 215-223
  - CV 63
- ## D
- D command 131, 144
  - data, transferring 41-42
  - data bits 137
  - data bus 213
  - Data Carrier Detect [60]
  - data format 137, 138, 144, 151
  - data inputs 21
  - Data Set Ready *See* DSR
  - Data Terminal Ready *See* DTR
  - DCB 261
  - DCD [60]
  - decimal, negative [107]
  - device signature 72
  - DEVNO [21]
  - DHIRES 49, 104, 106, 107, 166
  - diagnostics, built-in [62, 65]
  - differences among Apple II's [61-78]
  - disable MouseText 65
  - DISK 221, 222
  - disk
    - controller cards [74]
    - controller unit *See* IWM
    - input and output 124-126
    - I/O firmware entry points 20
  - disk drive 8
    - connector 252
    - port [50]
    - speed 13
  - disk-use light 6, [71]
  - display
    - address mapping 235-238
    - inverse 191
    - memory addressing 234
    - memory switches 43-47
    - modes 104-108, 239-247
    - normal 191
    - page maps 108-114
    - pages 102-103

- DISVBL 166
- DISXY 166
- DMA transfers [70]
- DOS 126, 130, 143, 179, 180, [39, 69]
  - interrupts [42]
  - zero page use [16-18]
- double-high-resolution 245
  - graphics [74]
  - colors 100-101
- drive, external, startup 126
- drive motor 49
- DSR 256, 261, [60]
- DSR1B 257
- DSR2B 257
- DTR 260
- DTR1B 257
- DTR2B 257
- dumb terminal 159
- Dvorak keyboard 6, [88]
- dynamic-RAM refreshment and timing 226-229

## E

- echo 131, 145, 155, 260
- EIA standard 258
- 80 columns 65, 93
- 80/40 column switch 5
- 80COL 104, 105, 107, 108, 218, 219, 220
- 80STORE 39, 44, 45, 104, 105, 107, 108, 216, 238, 241
- electrical power 206
- EN80 217
- enable MouseText 65
- ENBVBL 166
- ENBXY 166
- ENCLCRAM 216
- English keyboard [90]
- enhanced video firmware 20, 224
- enter terminal mode 145
- entry points, firmware [31-36]
- environmental specifications 205-206

- ESC** 4
- ESC** (4) 61
- ESC** (8) 61
- escape codes 60
- escape sequences 4
- even-parity [114]
- EXAMINE command 190
- examining memory contents 181
- examining registers 190
- expansion ROM space 73
- Extended 80-Column Text Card [64]
- external drive startup 126
- external interrupts [55]
- external power connector 207
- EXTINT 256, [55, 60]

## F

- FCC [99]
- firmware 12
  - entry points [30-36]
  - listings [126-215]
  - locations [30-36]
  - protocol 71, 134, 148
  - video routines 115-123
- flag inputs 21
- FLASH 256
- flashing characters 68
- flashing checkerboard cursor 55
- flashing power light 6
- forced cold start 50
- 14M 215, 220, 221
- FORTAN [41]
- 40 columns, switching to 80 5
- 40-column 65, 93
- 48K memory 34, 35, 39
- framing errors 258
- French keyboard [91-92]
- full duplex 156-158



## G

- GAME I/O connector [76]
- game input 267
- game paddles *See* hand controls
- GAMESW0 268
- GAMESW1 268
- General Logic Unit (GLU) 13
- German keyboard [93]
- GETLN 58-62, 180
- GETLN1 59, 82
- GETLNZ 82
- GLU 221
- GND 257
- GO command 189, 190, 192, 198
- graphic bits [109]
- graphics mode 96-102
- greater than sign (>) 59

## H

- half duplex 155
- hand control 8, 173-178
  - circuits 269
  - connector 174
  - input [76]
  - signals 270
- hand controller 267
- handle 9, 206
- hardware
  - accesses 21
  - addresses [66]
  - locations 181, [15]
  - page locations 164
- headphones 232
- heat 206
- hexadecimal [106]
  - arithmetic 193

- high-resolution 97
  - colors 243
  - display 243
  - double 245
  - graphics colors 98-99
  - Page 1 37
  - Page 2 38
- HIRES 44, 45, 104, 105, 107, 216, 218, [67]
- HLINE 117
- HOME 118
- HOMEMOUSE 168
- HRP1 37
- HRP1X 37, 45
- HRP2 45
- HRP2X 38
- humidity 205

## I

- I command 131, 145, 158
- I/O firmware, video routines 120-123
- I/O links 55
- icons 68
- identification bytes 71
- IEC [99]
- IN#2 143, 154
- IN#n 56, 70
- index registers 17
- INH 217
- INITMOUSE 169
- input and output, disk 124-126
- input buffer (page \$02) 36
- Input/Output Unit (IOU) 13, 215, 218-219, [78]
- instruction cycle times [63]
- Integer BASIC 59, [16-18, 41, 69]
- Integrated Woz Machine (IWM) 13
- internal converter 208
- internal voltage converter 206

- interrupt(s) 24, 75, 260, [40-60, 70]
  - ACIA [49]
  - Apple II and [42]
  - Apple II Plus and [42]
  - Apple IIe and [43]
  - disk drive port [49]
  - DOS and [42]
  - keyboard [52-53]
  - Monitor and [42]
  - mouse [49]
  - Pascal and [42]
  - 65C02 and [43]
  - 6551 [49]
  - vertical blanking [49]
- interrupt handler(s)
  - mouse 163
  - user's [57]
- interrupt requests 52
- interrupt vector [43-44]
- interrupt-handling sequence [45]
- inverse 65
  - characters 68
  - display 191
  - solid cursor 55
- INVERSE command 191
- invoking the monitor 179
- IOREST [36]
- IORTS [36]
- IOSAVE [36]
- IOU (Input/Output Unit) 13, 215, 218-219, [78]
- IOUDIS 49, 104, 106, 166, [67, 68]
- IOUSELIO 219
- IRQ 75, 156, 219, [43]
  - handling routine [34]
  - vector [36]
- ISO [84]
  - layout [89]
- Italian keyboard [94]
- IWM (Integrated Woz Machine) 13, 222

## J

- jack 7
- JMP \$C600 126
- JMP indirect instruction [3]
- joysticks *See* hand controls

## K

- K (1024) 17
- K command 131, 145
- KBD 217
- keyboard 229-231
  - buffer [52-53]
  - character decoder 225
  - circuit diagram 230
  - data [66]
  - input buffer 37
  - interrupts [52, 53]
  - layout [71]
    - ANSI [90]
    - British *See* English
    - Canadian [91-92]
    - Dvorak [88]
    - English [90]
    - French [91-92]
    - German [93]
    - ISO [90]
    - Italian [94-95]
    - Sholes [85]
    - Western Spanish [96]
  - signals 231
  - strobe 79, 229, [50, 66]
  - switch 5
  - standard 5
- KEYIN 55, 57, 58
- KSTRB 77, 219, 256
- KSW 56, 57, 70, 104

## L


- L command 131, 145
- LANGSW 256
- LDPS 220, 241, 251

- line feed 145, 152
  - automatic 131
- line length 136, 150
- line voltage 205
- line width 139, 144
- LIST command 199
- local 154
- low-resolution
  - colors 242
  - display 242
  - graphics 96
- M**
- machine identification [63]
- main memory screen holes 135-136, 149, 150
- main RAM 20
- MARK (1) 132
- MARK parity 138, [114]
- master clock 213
- maximum current drain 252
- memory
  - addressing 223-229
  - bank-switched 22
  - bus organization 224
  - comparing data in 188-189
  - display switches 43-47
  - dump 182-184
  - examining contents 181
  - 48K 34
  - map 18, [15-28]
  - moving data in 186-188
  - organization [64]
  - state [48]
  - switches, display 43-47
- Memory Management Unit *See* MMU
- microprocessor, 65C02 12, 15
- mini-phone jack 7
- MIXED 105, 107, 218, [67]
- mixed-modes displays 102
- MMU 13, 215-217, 267, 271, [78]
- mnemonic 199
- modem 8, 151
- modes, display 239-247
- monitor 8, 24, 59, 179-203, 224
  - entry point [36]
  - interrupts and [42]
  - output 248
  - register commands 189-190
  - ROM [69]
  - video routines 115
  - zero page use [15]
- mouse 8, 160-174, [49-50]
  - BASIC and 163, 172
  - Pascal and 171
  - button 171
    - interrupt mode 164
    - signals 266
  - clamping boundaries 171
  - connector 264
  - direction [59]
  - firmware 167
  - firmware entry points 20
  - hardware locations 164-167
  - input 262, [76]
  - interrupt handler 165
  - interrupts [58]
  - movement interrupt mode 163
  - operating modes 163
  - port 161-174
  - transparent mode 163
  - waveform 263
  - X direction 167
  - Y direction 167
- MOUSEID 264
- MouseText 65, 68-69, 90-91, [73, 114]
- MOUX1 167
- MOUY1 167
- MOVE command 186-188, 195, [36]
- MOVEAUX 41-42
- movement/button interrupt mode 164, 167
- movement interrupt mode 163, 167
- moving data in memory 186-188
- MSLOT [21]
- MSW 264

## N

- N command 131, 145, 156
- n CONTROL-K 56
- NE556 265, 271, [77]
- negative decimal [107]
- NEWIRQ [34]
- nibble [104]
- NMI vector [36, 43]
- non-maskable interrupts 52
- NORMAL command 191
- normal characters 65, 68
- normal display 191
- NTSC 87, 233, 242, 248, 251
- #6 130
- #7 143
- #8 143

## O

- odd-parity [114]
- old monitor ROM [62]
- 1 CONTROL-P 130
- 1VSOUND 251
- () 4, 82
- operand 199
- operating systems [39-40]
- operating temperature 205
- output and input, disk 124-126
- output jack 232

## P

- P command 132, 145
- P register 17
- paddle(s) 267
  - button 0 268
  - button 1 268
  - inputs [68, 76]
  - timing circuit [77]
- page 18
- page \$02 (input buffer) 36
- page \$03 36
- page \$04 36
- page \$08 37

- page 0 18
- page zero 24
- page 1 18
- PAGE2 44-45, 105, 107-108, 216,  
238, 241, [46-48, 67]
- page three [19]
- page 8, auxiliary RAM [52]
- PAL 233
- parity 145
  - bit(s) 138, 262
  - checking 260
- Pascal 67, 126, 130, 134, 170, [114]
  - ID byte 134, 148
  - interrupts and [42]
  - language [41]
  - operating system [40]
- PC (program counter) 16
- PCAS 220
- PDL0 176
- PDL0/XMOVE 219
- PDL1 176
- PEEK [40]
- peripheral identification numbers [112]
- peripheral-card memory space [65-66]
- peripheral-card ROM space [65]
- phone jack 7
- PIN numbers [112]
- PINIT 72, 121, 134, 148
- PLOT 118
- plotter 8
- POKE [40]
- ports 70, [70]
- POSMOUSE 168
- power 8
  - connector 207
  - consumption 207, 208
  - light 6, [71]
  - requirements 206
  - supply [100]
- power-on light [71]
- power-up byte 51

PR#1 130  
 PR#2 143, 154  
 PR#6 126  
 PR#n 56, 70  
 PRAS 217, 219, 220, 251  
 PRBL2 118  
 PRBYTE 118  
 PREAD 72, 121, 134, 148, 177  
 PRERR 118  
 PRHEX 118  
 primary character set 68, [73]  
 printer 8  
 PRINTER: 130  
 processor status register 17  
 ProDOS 126, 130, 143, 180, [39, 63]  
 program counter (PC) 16, 201  
 prompt 58, 154  
     characters 59  
 PRTAX 118  
 PSTATUS 72, 123, 134, 148  
 PTRIG 166  
 published entry points [32-36]  
*pull* from stack 17  
*push* onto stack 17  
 PWRITE 72, 121, 134, 148

## Q

Q command 145  
 Q3 215, 217, 219  
 question mark (?) 58, 59  
 quit terminal mode 145

## R

R command 132  
 R/W 217, 219, 221, 257  
 RA0-RA7 217  
 RAM 17  
     addressing 226-229  
     locations [15]  
 RAMRD 38, 39, 43, 44, 216, [46]  
 RAMWRT 38, 39, 43, 44, 216, [46]

random number 58  
 random-access memory (RAM) 17  
 RAS (row-address strobe) 228  
 RD1B 257  
 RD63 167  
 RD80COL 105  
 RD80STORE 105  
 RDALTCHAR 105  
 RDALTZP 26  
 RDBNK2 26  
 RDCHAR 82  
 RDCRAM [46]  
 RDDHIRES 106  
 RDHIRES 45, 105  
 RDIODIS 106, 166  
 RDKEY 55, 57  
 RDLGRAM 26  
 RDMIXED 105  
 RDPAGE2 105  
 RDRAMRD 39  
 RDRAMWRT 39  
 RDTXT 105  
 RDTNO 167  
 RDVBLMSK 166  
 RDXYMSK 166  
 RDYOEDGE 166  
 read-only memory (ROM) 17  
 READMOUSE 163, 168, [51-52]  
 receive register 262  
 registers 15, 213  
     examining 190  
 relative humidity 205  
 REMIN 143  
 remote 154, 159  
 remote device 145  
 REMOUT 143  
 (REPT) key [71]  
 Request to Send *See* RTS  
 (RESET) key 4, 79, 82, 221, 113, 256,  
     [71]  
 reset port 1 132  
 reset port 2 145  
 reset routine 48  
 reset vector 49-51, [36]  
 (RETURN) [84]




- retype 62
- RF modulator 233
- RGB monitor 245
- rollover 3
- ROM 17
- ROM addressing 224-225
- ROMEN2 217
- RS-232 129
- RSTVBL 166
- RSTXINT 166, 216
- RSTXY 166
- RSTYINT 166, 216
- RTS instruction 260, [36]


## S

- S command 132
- S register 17
- safety instructions 207, [99]
- schematic diagrams 271-276
- scratch-pad RAM [65]
- screen holes 36, 73, 74, 133, 134, 136, 149, 171-173, [20-22, 47]
- SCRN 119
- scroll 65
- SEGA 218
- SEGB 218, 220, 251
- self-tests *See* diagnostics, built-in
- SER 221, 256
- serial buffering [55]
- serial data transfer [57]
- serial firmware [50]
- serial I/O buffers [75]
- serial I/O port 128-159
- serial input buffer 37
- Serial Interface Card [74]
- serial interrupts [55, 56]
- serial port circuits 254
- serial port 1 20, 129-139
- serial port 2 20, 141-159
  - command character 143, 146
  - command character hardware
    - locations 130, 132, 134
  - firmware protocol 147
  - hardware locations 148
  - initial characteristics 130, 147
- SEROUT 251
- SERVENOUSE 163, 168, [51]
- SETCOL 119
- SETMOUSE 167-168, [50-51]
- SETPWRC 51
- 7M 220, 223
- (SHIFT) key 79, 229, [84]
- shift-key mod [68]
- Sholes keyboard 5
- signature byte 134, 148, 170
- simplified keyboard (Dvorak) [88]
- (6) 126
- 65C02 12, 15, [63]
  - address bus 213
  - addressing modes [10]
  - block diagram 211
  - clock 211
  - cycle time [1, 2]
  - data bus 213
  - data sheet [5-13]
  - differences from 6502 211, [1-3, 6-7]
  - execution time [1-2]
  - instruction set [12-13]
  - opcodes [12]
  - registers 213
  - signal descriptions [11]
  - timing diagram [8]
  - timing signals 214-215
- 6502 versus 65C02 211
- 6551 Asynchronous Communication
  - Interface Adapters *See* ACIA
- slot 7 drive 1 [74]
- SLOT3ROM [66]
- SLOTXROM [66]



- slots 70
  - versus ports [70]
- soft switches 22, 215, 218, 221
-  82
- SPACE (0) 132
- SPACE parity 138, [114]
- speaker 83-84, [67]
  - external 7
  - output jack 232
  - volume control 232
- SPKR 219
- stack 24, [42, 46]
- stack pointer 17
- standard I/O links 55
- standard keyboard 5
- start bit 137
- status register 134, 148, 261
- stop bits 137
- stop-list 65
- STORE command 194
- strobe 79
  - inputs 21
- SUD *See* System Utilities Disk
- Super Serial Cards [74]
- SW0 175
- SW1 175
- switch inputs 175, [76]
- switches, soft 22, 215
- SYNC 219, 233, 251
- system clock 213
- system monitor 179-203
- System Utilities Disk* 129, 131, 136, 141, 145, 150, [75, 112]

## T

- T command 145, 154-156, 159
-  [84]
- TD1B 257
- telephone jack 7
- temperature 205, 208

- terminal mode 145, [53]
- TEXT 105, 107, 218, 220, 221, 251, [67]
- text
  - and low-resolution graphics Page 1 36
  - and low-resolution Page 1X 36
  - and screen low-resolution Page 2 37
  - displays 241
  - modes 90-95
  - window 63, 66
- TLP1 36
- TLP1X 36, 45
- TLP2X 37
- toggle switches 22
- transferring control 42-43
- transferring data 41-42
- transmit/receive data register 134, 148
- transmit register 262
- transparent mode 163, 167, 171
- triggering paddle timers [68]

## U

- USA standard keyboard 5
- USER command 197
- user's interrupt handler [57]
- utility strobe [67]

## V

- validity check 49
- VBL [67, 73, 76]
- VBLINT 163, 164, 218, [67, 73]
- VDE [99]
- vectors 55
- ventilation 206
- VERIFY command 188, 196, [36]
- vertical blanking 163, [49, 50, 73]
  - interrupts [68]

VID 248  
VID7M 215, 220  
video  
    counters 233-234  
    display 225  
    display circuits 240  
    display modes 239-247  
    expansion 8  
    expansion connector 249-252  
    expansion output 249  
    output signals 248  
    routines  
        firmware 115-123  
        I/O firmware 120-123  
        monitor 115-119  
VLINE 119  
voltage 205  
    converter 10  
volume control 7, 232

## W

WAIT [36]  
warm-start procedure 50  
Western Spanish keyboard [96]  
WNDW 219, 233, 251  
word [106]  
Woz Integrated Machine 13, 222

## X

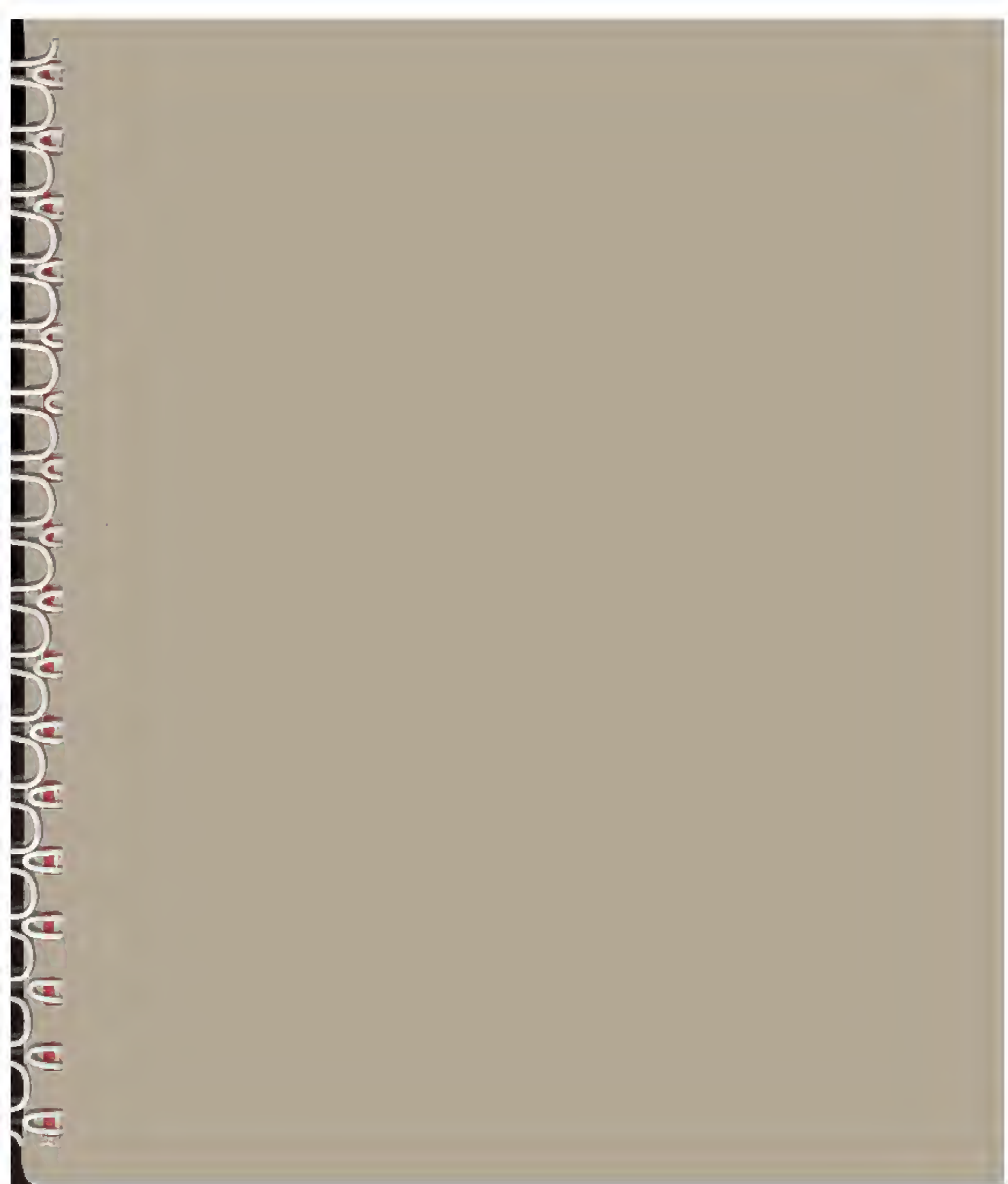
X register 17  
X0 215, 218, 262, 264  
X1 215, 263, 264  
XFER 41, 42  
XINT 164, [66, 67]  
XOEDGE 166

## Y

Y register 17  
Y0 218, 262, 264  
Y1 263, 264  
YINT 164, [66, 67]  
YMOVE 219  
YOEDGE 166

## Z

Z command 132, 139  
zap 132, 139, 145  
zero page 24, 184





Apple Computer, Inc.  
20525 Mariani Avenue  
Cupertino, California 95014  
(408) 996-1010  
TLX 171-576

030-0814-A  
1984 Apple Computer, Inc.  
Printed in U.S.A.